SDPMflut User's guide, version 0.72

G. Dimitriadis

December 2025

Contents

1 Introduction

`SDPMflut` is a Python package to perform subsonic compressible aerodynamic and aeroelastic analysis on wings and bodies using the compressible unsteady Source and Doublet Panel Method (SDPM) and the compressible unsteady Doublet Lattice Method. It is largely based on Unsteady Aerodynamics: Potential and Vortex Methods by G. Dimitriadis [1] and its core components are similar to those included in the companion site of the textbook, albeit written in Python instead of Matlab. The present unsteady aerodynamic formulation of the SDPM is also described in [2, 3], while the aeroelastic formulation of the SDPM is presented in [4, 5, 6]. The nonplanar quadratic DLM included in `SDPMflut` was developed from the descriptions in [7, 8, 9, 10].

2 Source and Doublet Lattice Method

Under inviscid, irrotational and isentropic flow assumptions and for small perturbations the continuity equation simplifies to the linearized small disturbance equation

$$(1 - M_\infty^2)\phi_{xx} + \phi_{yy} + \phi_{zz} - \frac{2M_\infty}{a_\infty}\phi_{xt} - \frac{1}{a_\infty^2}\phi_{tt} = 0 \tag{1}$$

where $M_\infty = Q_\infty/a_\infty$ is the free stream Mach number, Q_∞ is the free stream airspeed along the x direction, a_∞ is the free stream speed of sound and ϕ is the velocity perturbation potential, defined from

$$u = \frac{\partial \phi}{\partial x}, \;\; v = \frac{\partial \phi}{\partial y}, \;\; w = \frac{\partial \phi}{\partial z}$$

and u, v, w are the perturbation velocities in the x, y, z directions. Under the same assumptions, the momentum equations can be integrated and expanded up to second order, such that

$$c_p = 1 - \frac{Q^2}{Q_\infty^2} + \frac{M_\infty^2}{Q_\infty^2}\phi_x^2 - \frac{2}{Q_\infty^2}\phi_t + \frac{M_\infty^2}{Q_\infty^4}\phi_t^2 + \frac{2M_\infty^2}{Q_\infty^3}\phi_x\phi_t \tag{2}$$

where c_p is the pressure coefficient

$$c_p = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty Q_\infty^2}$$

where p is the pressure at any point in the flow and p_∞ is the free stream pressure, $Q^2 = u^2 + v^2 + w^2$ and ρ_∞ the free stream density.

The present source and doublet panel method solves equation 1 for the velocity potential $\phi(t)$ on the surface of a wing or other body and then calculates the pressure coefficient on the surface from equation 2. Both solutions are evaluated after applying the Fourier and Prandtl-Glauert transformations. The latter consists in expressing the geometry in coordinates ξ, η, ζ, such that

$$\xi = x/\beta, \ \eta = y, \ \zeta = z \tag{3}$$

where $\beta^2 = 1 - M_\infty^2$. Furthermore, the quasi-fixed geometry assumption is enforced, such that:

- all deformations of all bodies are small so that their geometry can be approximated as constant in time.
- the deformations are represented mathematically by means of the relative velocity between the fluid and the surface.

All the assumptions mentioned above mean that the SDPM is a fast and accurate aerodynamic modelling method as long as structural deformations are small, there is no significant flow separation and there are no shock waves in contact with the surface. Furthermore, as the present formulation of the SDPM is purely subsonic, the free stream Mach number must be less than one. Extensions to supersonic and transonic flow conditions are under development.

2.1 Calculating the potential on the surface

The fundamental equation of the unsteady compressible Source and Doublet Panel Method (SDPM) is Green's theorem in the frequency domain [11, 12]

$$\left(\hat{\boldsymbol{B}}_\phi(k) - \frac{1}{2}\boldsymbol{I}\right)\boldsymbol{\mu}(k) + \hat{\boldsymbol{C}}_\phi(k)\boldsymbol{\mu}_w(k) = -\hat{\boldsymbol{A}}_\phi(k)\boldsymbol{\sigma}(k) \tag{4}$$

Equation 4 is written out for the N panels on the surface of a wing and the N_w panels on the flat wake, such that $k = \omega c_0/2Q_\infty$ is the reduced frequency, ω is the frequency in rad/s, c_0 is a reference chord length, $\hat{\boldsymbol{A}}_\phi(k)$, $\hat{\boldsymbol{B}}_\phi(k)$ are $N \times N$ unsteady source and doublet influence coefficient matrices, $\hat{\boldsymbol{C}}_\phi(k)$ is the $N \times N_w$ unsteady doublet influence coefficient matrix of the wake on the wing, $\boldsymbol{\sigma}(k)$, $\boldsymbol{\mu}(k)$ are $N \times 1$ vectors of the source and doublet strengths on the wing panels and $\boldsymbol{\mu}_w(k)$ is the $N_w \times 1$ vector of the doublet strengths on the wake panels. The doublet strength on the surface is equal to the unknown potential on the surface, the source strength on the surface is determined from the impermeability boundary condition and the doublet strength in the wake is determined from the Kutta condition

$$\boldsymbol{\mu}_w(k) = \boldsymbol{P}_e(k)\boldsymbol{P}_c\boldsymbol{\mu}(k) \tag{5}$$

where

$$\begin{aligned} \boldsymbol{P}_c &= \begin{pmatrix}\boldsymbol{0}_{n\times(2m-1)n} & \boldsymbol{I}_n\end{pmatrix} - \begin{pmatrix}\boldsymbol{I}_n & \boldsymbol{0}_{n\times(2m-1)n}\end{pmatrix} & (6) \\ \boldsymbol{P}_e(k) &= \begin{pmatrix} \boldsymbol{I}_n \mathrm{e}^{-\mathrm{i}2k/m} \\ \boldsymbol{I}_n \mathrm{e}^{-\mathrm{i}4k/m} \\ \vdots \\ \boldsymbol{I}_n \mathrm{e}^{-\mathrm{i}2m_w k/m} \end{pmatrix} & (7) \end{aligned}$$

Applying these two conditions leads to the solution for the doublet strength on the surface

$$\boldsymbol{\mu}(k) = \boldsymbol{\phi}(k) = -\boldsymbol{K}(k)\boldsymbol{\mu}_n(k) \tag{8}$$

where

$$\begin{aligned} \boldsymbol{K}(k) &= \left(\frac{2\mathrm{i}kM_\infty^2}{c_0\beta}\hat{\boldsymbol{A}}_\phi(k)\circ\boldsymbol{n}_\xi + \hat{\boldsymbol{B}}_\phi(k) - \frac{1}{2}\boldsymbol{I} + \hat{\boldsymbol{C}}_\phi(k)\boldsymbol{P}_e(k)\boldsymbol{P}_c\right)^{-1}\hat{\boldsymbol{A}}_\phi(k) \\ \boldsymbol{\mu}_n(k) &= -\left(\frac{1}{\beta}\boldsymbol{u}_m(k)\circ\boldsymbol{n}_\xi + \boldsymbol{v}_m(k)\circ\boldsymbol{n}_\eta + \boldsymbol{w}_m(k)\circ\boldsymbol{n}_\zeta\right) & (9) \end{aligned}$$

$\boldsymbol{n}_\xi$, $\boldsymbol{n}_\eta$, $\boldsymbol{n}_\zeta$ are the three components of the unit vectors normal to the panels and pointing outwards written in Prandtl-Glauert coordinates, and the Hadamard operator $\circ$ is used to denote the element-by-element multiplication of vectors or of each of the columns of a matrix by the same column vector. It should be noted that $\boldsymbol{K}(k)$ has dimensions of meters while $\boldsymbol{\mu}_n(k)$ has dimensions of m/s, such that $\boldsymbol{\mu}(k)$ has dimensions of m^2/s.

Once the doublet strengths on the surface, $\boldsymbol{\mu}(k)$, have been evaluated, the next step is to calculate the perturbation velocities on the surface by numerically differentiating $\boldsymbol{\mu}(k)$. The result is

$$\boldsymbol{\phi}_x(k) = \boldsymbol{K}_x(k)\boldsymbol{\mu}_n(k),\ \ \boldsymbol{\phi}_y(k) = \boldsymbol{K}_y(k)\boldsymbol{\mu}_n(k),\ \ \boldsymbol{\phi}_z(k) = \boldsymbol{K}_z(k)\boldsymbol{\mu}_n(k) \tag{10}$$

where $\boldsymbol{K}_x(k)$, $\boldsymbol{K}_y(k)$, $\boldsymbol{K}_z(k)$ are non-dimensional functions of the geometry and finite difference matrices and are detailed in [3, 4]. The total flow velocities on the surface are given by

$$\boldsymbol{u}(k) = \boldsymbol{u}_m(k) + \boldsymbol{\phi}_x(k),\ \ \boldsymbol{v}(k) = \boldsymbol{v}_m(k) + \boldsymbol{\phi}_y(k),\ \ \boldsymbol{w}(k) = \boldsymbol{w}_m(k) + \boldsymbol{\phi}_z(k) \tag{11}$$

where $\boldsymbol{u}_m(k)$, $\boldsymbol{v}_m(k)$, $\boldsymbol{w}_m(k)$ are the relative velocities between the flow and the surface due to the motion.

The steady solution is obtained for $k = 0$, such that equation 8 becomes

$$\boldsymbol{\mu}(0) = \boldsymbol{\phi}(0) = -\boldsymbol{K}(0)\boldsymbol{\mu}_n(0) \tag{12}$$

where

$$\begin{aligned} \boldsymbol{K}(0) &= \left(\boldsymbol{B}_\phi - \frac{1}{2}\boldsymbol{I} + \boldsymbol{C}_\phi\boldsymbol{P}_e(0)\boldsymbol{P}_c\right)^{-1}\boldsymbol{A}_\phi & (13) \\ \boldsymbol{\mu}_n(0) &= -\left(\frac{1}{\beta}U_\infty\boldsymbol{n}_\xi + V_\infty\boldsymbol{n}_\eta + W_\infty\boldsymbol{n}_\zeta\right) & (14) \end{aligned}$$

$$\boldsymbol{P}_e(0) = \begin{pmatrix} \boldsymbol{I}_n \\ \boldsymbol{I}_n \\ \vdots \\ \boldsymbol{I}_n \end{pmatrix} \tag{15}$$

$$\boldsymbol{\phi}_x(0) = \boldsymbol{K}_x(0)\boldsymbol{\mu}_n(0),\ \boldsymbol{\phi}_y(0) = \boldsymbol{K}_y(0)\boldsymbol{\mu}_n(0),\ \boldsymbol{\phi}_z(0) = \boldsymbol{K}_z(0)\boldsymbol{\mu}_n(0) \tag{16}$$

$$\boldsymbol{u}(0) = U_\infty + \boldsymbol{\phi}_x(0),\ \boldsymbol{v}(0) = V_\infty + \boldsymbol{\phi}_y(0),\ \boldsymbol{w}(0) = W_\infty + \boldsymbol{\phi}_z(0) \tag{17}$$

with $Q_\infty^2 = U_\infty^2 + V_\infty^2 + W_\infty^2$, $\boldsymbol{A}_\phi$, $\boldsymbol{B}_\phi$, $\boldsymbol{C}_\phi$ being the steady influence coefficient matrices. Note that the form of equation 1 implies that $V_\infty << U_\infty$ and $W_\infty << U_\infty$, such that only U_∞ is comparable to a_∞. In other words, the direction of compressibility is the x axis.

2.2 Calculating the pressure on the surface

Appendix A.1 shows that, after substituting equations 8 to 17 is the Fourier transform of equation 2, the pressure coefficient on the panel control points is given by

$$\begin{aligned} \boldsymbol{c}_p(k) &= -\frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\xi \circ \bar{\boldsymbol{u}}_m(k)) - 2\bar{\boldsymbol{u}}(0) \circ \bar{\boldsymbol{u}}_m(k) - \bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\eta \circ \bar{\boldsymbol{v}}_m(k)) - 2\bar{\boldsymbol{v}}(0) \circ \bar{\boldsymbol{v}}_m(k) \\ &\quad -\bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\zeta \circ \bar{\boldsymbol{w}}_m(k)) - 2\bar{\boldsymbol{w}}(0) \circ \bar{\boldsymbol{w}}_m(k) - \frac{2\mathrm{i}k}{c_0\beta}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\xi \circ \bar{\boldsymbol{u}}_m(k)) \\ &\quad -\frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\eta \circ \bar{\boldsymbol{v}}_m(k)) - \frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\zeta \circ \bar{\boldsymbol{w}}_m(k)) \end{aligned} \tag{18}$$

where

$$\begin{aligned} \bar{\boldsymbol{C}}_0(k) &= -2\boldsymbol{K}_x(k) \circ \bar{\boldsymbol{u}}(0) - 2\boldsymbol{K}_y(k) \circ \bar{\boldsymbol{v}}(0) - 2\boldsymbol{K}_z(k) \circ \bar{\boldsymbol{w}}(0) + 2M_\infty^2 \boldsymbol{K}_x(k) \circ \bar{\boldsymbol{\phi}}_x(0) \\ \bar{\boldsymbol{C}}_1(k) &= 2\boldsymbol{K}(k) - 2M_\infty^2 \boldsymbol{K}(k) \circ \bar{\boldsymbol{\phi}}_x(0) \end{aligned} \tag{19}$$

and the overbars denote the nornalized potential and velocities $\bar{\boldsymbol{\phi}}(k) = \boldsymbol{\phi}(k)/Q_\infty$, $\bar{\boldsymbol{u}}(k) = \boldsymbol{u}(k)/Q_\infty$, $\bar{\boldsymbol{v}}(k) = \boldsymbol{v}(k)/Q_\infty$, $\bar{\boldsymbol{w}}(k) = \boldsymbol{w}(k)/Q_\infty$, $\bar{\boldsymbol{u}}_m(k) = \boldsymbol{u}_m(k)/Q_\infty$, $\bar{\boldsymbol{v}}_m(k) = \boldsymbol{v}_m(k)/Q_\infty$, $\bar{\boldsymbol{w}}_m(k) = \boldsymbol{w}_m(k)/Q_\infty$, $\bar{\boldsymbol{\phi}}_x(k) = \boldsymbol{\phi}_x(k)/Q_\infty$, $\bar{\boldsymbol{\phi}}_y(k) = \boldsymbol{\phi}_y(k)/Q_\infty$, $\bar{\boldsymbol{\phi}}_z(k) = \boldsymbol{\phi}_z(k)/Q_\infty$, $\bar{U}_\infty = U_\infty/Q_\infty$, $\bar{V}_\infty = V_\infty/Q_\infty$, $\bar{W}_\infty = W_\infty/Q_\infty$. It should be noted that both $\bar{\boldsymbol{\phi}}(k)$ and $\bar{\boldsymbol{C}}_1(k)$ have dimensions of meters, such that $\boldsymbol{c}_p(k)$ is non-dimensional. Substituting the same normalized quantities into equation 59, such that the steady pressure coefficient becomes

$$\boldsymbol{c}_{p_0} = 1 - (\bar{\boldsymbol{u}}(0) \circ \bar{\boldsymbol{u}}(0) + \bar{\boldsymbol{v}}(0) \circ \bar{\boldsymbol{v}}(0) + \bar{\boldsymbol{w}}(0) \circ \bar{\boldsymbol{w}}(0)) + M_\infty^2 \left(\bar{\boldsymbol{\phi}}_x(0) \circ \bar{\boldsymbol{\phi}}_x(0)\right) \tag{20}$$

Equations 20 and 18 are the required steady and unsteady pressure distributions around the surface. In order to calculated them, the free stream and relative motion between the flow and the surface must be specified. The free stream is straightforward; it depends on the prescribed flight condition, which are usually described by the angle of attack α_0 and the angle of sideslip β_0, such that

$$\bar{U}_\infty = \cos\alpha_0 \cos\beta_0,\ \bar{V}_\infty = -\sin\beta_0, \bar{W}_\infty = \sin\alpha_0 \cos\beta_0 \tag{21}$$

The relative velocity due to the unsteady motion at frequency k can be rigid and/or flexible. For rigid motion around the body's centre of gravity, having set the latter as the origin of the coordinate system, the linearized motion-induced velocities are given by

$$\begin{aligned}
\bar{\boldsymbol{u}}_m(k) &= \bar{V}_\infty\psi(k) - \bar{W}_\infty\theta(k) - \bar{q}(k)\boldsymbol{z}_c + \bar{r}(k)\boldsymbol{y}_c - \bar{u}(k) \\
\bar{\boldsymbol{v}}_m(k) &= -\bar{U}_\infty\psi(k) + \bar{W}_\infty\phi(k) + \bar{p}(k)\boldsymbol{z}_c - \bar{r}(k)\boldsymbol{x}_c - \bar{v}(k)(k) \\
\bar{\boldsymbol{w}}_m(k) &= \bar{U}_\infty\theta(k) - \bar{V}_\infty\phi(k) + \bar{q}(k)\boldsymbol{x}_c - \bar{p}(k)\boldsymbol{y}_c - \bar{w}(k)
\end{aligned} \tag{22}$$

where $\boldsymbol{x}_c$, $\boldsymbol{y}_c$, $\boldsymbol{z}_c$ are the coordinates of the panel control points, $u(k)$, $v(k)$, $w(k)$ are the rigid-body translation velocities, $p(k)$, $q(k)$, $r(k)$ are the rigid-body roll, pitch and yaw velocities, $\phi(k)$, $\theta(k)$, $\psi(k)$ are the roll, pitch and yaw angles and $\bar{u}(k) = u(k)/Q_\infty$, $\bar{v}(k) = v(k)/Q_\infty$, $\bar{w}(k) = w(k)/Q_\infty$, $\bar{p}(k) = p(k)/Q_\infty$, $\bar{q}(k) = q(k)/Q_\infty$, $\bar{r}(k) = r(k)/Q_\infty$. Note that $\bar{p}(k)$, $\bar{q}(k)$, $\bar{r}(k)$ have dimensions of 1/m so that $\bar{\boldsymbol{u}}_m(k)$, $\bar{\boldsymbol{v}}_m(k)$, $\bar{\boldsymbol{w}}_m(k)$, are non-dimensional.

Flexible deformations are modelled in `SDPMflut` using a modal description. The body's deformation is written as

$$\boldsymbol{x}_{FE}(t) = \boldsymbol{\Phi}\boldsymbol{q}(t)$$

where $\boldsymbol{x}_{FE}(t)$ is the $6N_{FE} \times 1$ vector of degrees of freedom of a finite element model, N_{FE} is the number of nodes in the model, $\boldsymbol{\Phi}$ is a $6N_{FE} \times K$ matrix of mode shapes, K is the number of retained modes and $\boldsymbol{q}(t)$ is a $K \times 1$ vector of generalized coordinates. The mode shape matrix $\boldsymbol{\Phi}$ contains translations in the x, y, z directions and rotations around the x, y, z axes. Then, it can be decomposed as

$$\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\Phi}_x \\ \boldsymbol{\Phi}_y \\ \boldsymbol{\Phi}_z \\ \boldsymbol{\Phi}_\phi \\ \boldsymbol{\Phi}_\theta \\ \boldsymbol{\Phi}_\psi \end{pmatrix}$$

where $\boldsymbol{\Phi}_x$, $\boldsymbol{\Phi}_y$, $\boldsymbol{\Phi}_z$ are $N_{FE} \times K$ translation mode shapes and $\boldsymbol{\Phi}_\phi$, $\boldsymbol{\Phi}_\theta$, $\boldsymbol{\Phi}_\psi$ are $N_{FE} \times K$ rotation mode shapes in the roll, pitch and yaw directions respectively. The mode shapes of the finite element model must be interpolated onto the control points of the SDPM grid, such that they become $N \times K$ matrices $\tilde{\boldsymbol{\Phi}}_x$, $\tilde{\boldsymbol{\Phi}}_y$, $\tilde{\boldsymbol{\Phi}}_z$, $\tilde{\boldsymbol{\Phi}}_\phi$, $\tilde{\boldsymbol{\Phi}}_\theta$, $\tilde{\boldsymbol{\Phi}}_\psi$, the tilde denoting interpolated quantities and N being the number of SDPM panels. Consequently, the motion-induced velocities are given by

$$\begin{aligned}
\bar{\boldsymbol{u}}_m(k) &= \bar{V}_\infty\tilde{\boldsymbol{\Phi}}_\psi\boldsymbol{q}(k) - \bar{W}_\infty\tilde{\boldsymbol{\Phi}}_\theta\boldsymbol{q}(k) - \frac{2\mathrm{i}k}{c_0}\tilde{\boldsymbol{\Phi}}_x\boldsymbol{q}(k) \\
\bar{\boldsymbol{v}}_m(k) &= -\bar{U}_\infty\tilde{\boldsymbol{\Phi}}_\psi\boldsymbol{q}(k) + \bar{W}_\infty\tilde{\boldsymbol{\Phi}}_\phi\boldsymbol{q}(k) - \frac{2\mathrm{i}k}{c_0}\tilde{\boldsymbol{\Phi}}_y\boldsymbol{q}(k) \\
\bar{\boldsymbol{w}}_m(k) &= \bar{U}_\infty\tilde{\boldsymbol{\Phi}}_\theta\boldsymbol{q}(k) - \bar{V}_\infty\tilde{\boldsymbol{\Phi}}_\phi\boldsymbol{q}(k) - \frac{2\mathrm{i}k}{c_0}\tilde{\boldsymbol{\Phi}}_z\boldsymbol{q}(k)
\end{aligned} \tag{23}$$

Substituting equations 22 and/or 23 into expression 18 results in the numerical values of the oscillatory pressure distribution at frequency k. Note that $\boldsymbol{c}_p(k)$ is a complex quantity. Note that $u(k)$, $v(k)$, $w(k)$, $p(k)$, $q(k)$, $r(k)$ and $\boldsymbol{q}(k)$ are known if the motion is prescribed. If the motion is free, the pressure distribution can be written in the form of pressure derivatives. Then, the pressure distribution due to the rigid motion is written as

$$\begin{aligned}
\boldsymbol{c}_p(k) \quad = \quad & \boldsymbol{c}_{p_u}(k)u(k) + \boldsymbol{c}_{p_v}(k)v(k) + \boldsymbol{c}_{p_w}(k)w(k) + \boldsymbol{c}_{p_\theta}(k)\theta(k) \\
& + \boldsymbol{c}_{p_\psi}(k)\psi(k) + \boldsymbol{c}_{p_p}(k)p(k) + \boldsymbol{c}_{p_q}(k)q(k) + \boldsymbol{c}_{p_r}(k)r(k) \\
& + \boldsymbol{c}_{p_{\dot{u}}}(k)u(k) + \mathrm{i}k\boldsymbol{c}_{p_{\dot{v}}}(k)v(k) + \mathrm{i}k\boldsymbol{c}_{p_{\dot{w}}}(k)w(k) + \mathrm{i}k\boldsymbol{c}_{p_{\dot{p}}}(k)p(k) \\
& + \mathrm{i}k\boldsymbol{c}_{p_{\dot{q}}}(k)q(k) + \mathrm{i}k\boldsymbol{c}_{p_{\dot{r}}}(k)r(k)
\end{aligned} \tag{24}$$

and `SDPMflut` calculates the $N \times 1$ pressure derivative vectors $\boldsymbol{c}_{p_u}(k)$, $\boldsymbol{c}_{p_v}(k)$, etc, expressions for which are given in appendix A.2. For flexible motion, the pressure distribution is written as

$$\begin{aligned}
\boldsymbol{c}_p(k) \quad = \quad & \left(\boldsymbol{c}_{p_\phi}(k) + \boldsymbol{c}_{p_\theta}(k) + \boldsymbol{c}_{p_\psi}(k)\right)\boldsymbol{q}(k) \\
& + \mathrm{i}k\left(\boldsymbol{c}_{p_{\dot{x}}}(k) + \boldsymbol{c}_{p_{\dot{y}}}(k) + \boldsymbol{c}_{p_{\dot{z}}}(k) + \boldsymbol{c}_{p_{\dot{\phi}}}(k) + \boldsymbol{c}_{p_{\dot{\theta}}}(k) + \boldsymbol{c}_{p_{\dot{\psi}}}(k)\right)\boldsymbol{q}(k) \\
& + (\mathrm{i}k)^2\left(\boldsymbol{c}_{p_{\ddot{x}}}(k) + \boldsymbol{c}_{p_{\ddot{y}}}(k) + \boldsymbol{c}_{p_{\ddot{z}}}(k)\right)\boldsymbol{q}(k)
\end{aligned} \tag{25}$$

and `SDPMflut` calculates the $N \times K$ pressure derivative matrices $\boldsymbol{c}_{p_\phi}(k)$, $\boldsymbol{c}_{p_\theta}(k)$, $\boldsymbol{c}_{p_\psi}(k)$ etc, expressions for which are given in appendix A.4.

2.3 Calculating the aerodynamic loads

The steady aerodynamic loads acting on each panel are given by

$$\begin{aligned}
\boldsymbol{F}_x(0) &= -\boldsymbol{c}_{p_0} \circ \boldsymbol{s} \circ \boldsymbol{n}_x \\
\boldsymbol{F}_y(0) &= -\boldsymbol{c}_{p_0} \circ \boldsymbol{s} \circ \boldsymbol{n}_y \\
\boldsymbol{F}_z(0) &= -\boldsymbol{c}_{p_0} \circ \boldsymbol{s} \circ \boldsymbol{n}_z \\
\boldsymbol{M}_x(0) &= (\boldsymbol{y}_c - y_{f0})\boldsymbol{F}_z(0) - (\boldsymbol{z}_c - z_{f0})\boldsymbol{F}_y(0) \\
\boldsymbol{M}_x(0) &= -(\boldsymbol{x}_c - x_{f0})\boldsymbol{F}_z(0) + (\boldsymbol{z}_c - z_{f0})\boldsymbol{F}_x(0) \\
\boldsymbol{M}_x(0) &= (\boldsymbol{x}_c - x_{f0})\boldsymbol{F}_y(0) - (\boldsymbol{y}_c - y_{f0})\boldsymbol{F}_x(0)
\end{aligned} \tag{26}$$

where $\boldsymbol{F}_x(0)$, $\boldsymbol{F}_y(0)$, $\boldsymbol{F}_z(0)$ are $N \times 1$ vectors of aerodynamic force per Pascal, $\boldsymbol{M}_x(0)$, $\boldsymbol{M}_y(0)$, $\boldsymbol{M}_z(0)$ are $N \times 1$ vectors of aerodynamic moment per Pascal around point (x_{f0}, x_{f0}, x_{f0}), $\boldsymbol{n}_x$, $\boldsymbol{n}_x$, $\boldsymbol{n}_x$, are the $N \times 1$ components of the unit vectors normal to the panels in cartesian coordinates, $\boldsymbol{s}$ is the $N \times 1$ vector of the areas of the panels, $\boldsymbol{c}_{p_0}$ is calculated from equation 20. The negative signs in these equations are due to the fact that the unit vectors normal to the surfaces are pointing outwards. Similarly, the unsteady aerodynamic loads at k are calculated from

$$\begin{aligned}
\boldsymbol{F}_x(k) &= -\boldsymbol{c}_p(k) \circ \boldsymbol{s} \circ \boldsymbol{n}_x \\
\boldsymbol{F}_y(k) &= -\boldsymbol{c}_p(k) \circ \boldsymbol{s} \circ \boldsymbol{n}_y \\
\boldsymbol{F}_z(k) &= -\boldsymbol{c}_p(k) \circ \boldsymbol{s} \circ \boldsymbol{n}_z
\end{aligned} \tag{27}$$

$$
\begin{aligned}
\boldsymbol{M}_x(k) &= (\boldsymbol{y}_c - y_{f_0})\boldsymbol{F}_z(k) - (\boldsymbol{z}_c - z_{f_0})\boldsymbol{F}_y(k) \\
\boldsymbol{M}_x(k) &= -(\boldsymbol{x}_c - x_{f_0})\boldsymbol{F}_z(k) + (\boldsymbol{z}_c - z_{f_0})\boldsymbol{F}_x(k) \\
\boldsymbol{M}_x(k) &= (\boldsymbol{x}_c - x_{f_0})\boldsymbol{F}_y(k) - (\boldsymbol{y}_c - y_{f_0})\boldsymbol{F}_x(k)
\end{aligned}
$$

where $\boldsymbol{c}_p(k)$ is calculated from equation 24 or 25.

2.4 Calculating the flutter solution

The aeroelastic equation for flexible motion is given by

$$
\boldsymbol{A}\ddot{\mathbf{q}} + \boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{E}\mathbf{q} = \frac{1}{2}\rho_\infty Q_\infty^2 \left(\boldsymbol{Q}_0(k) + \mathrm{i}k\boldsymbol{Q}_1(k) + (\mathrm{i}k)^2\boldsymbol{Q}_2(k)\right)\mathbf{q} \tag{28}
$$

where $\boldsymbol{A}$, $\boldsymbol{C}$, $\boldsymbol{E}$ are $K \times K$ structural modal mass, damping and stiffness matrices and $\boldsymbol{Q}_0(k)$, $\boldsymbol{Q}_1(k)$, $\boldsymbol{Q}_2(k)$ are generalized aerodynamic stiffness, damping and mass matrices, given by

$$
\begin{aligned}
\boldsymbol{Q}_0(k) &= \left(\tilde{\boldsymbol{\Phi}}_x^T\boldsymbol{F}_{x_0}(k) + \tilde{\boldsymbol{\Phi}}_y^T\boldsymbol{F}_{y_0}(k) + \tilde{\boldsymbol{\Phi}}_z^T\boldsymbol{F}_{z_0}(k)\right) \\
\boldsymbol{Q}_1(k) &= \left(\tilde{\boldsymbol{\Phi}}_x^T\boldsymbol{F}_{x_1}(k) + \tilde{\boldsymbol{\Phi}}_y^T\boldsymbol{F}_{y_1}(k) + \tilde{\boldsymbol{\Phi}}_z^T\boldsymbol{F}_{z_1}(k)\right) \\
\boldsymbol{Q}_2(k) &= \left(\tilde{\boldsymbol{\Phi}}_x^T\boldsymbol{F}_{x_2}(k) + \tilde{\boldsymbol{\Phi}}_y^T\boldsymbol{F}_{y_2}(k) + \tilde{\boldsymbol{\Phi}}_z^T\boldsymbol{F}_{z_2}(k)\right)
\end{aligned} \tag{29}
$$

where

$$
\begin{aligned}
\boldsymbol{F}_{x_0}(k) &= -\left(\boldsymbol{c}_{p_\phi}(k) + \boldsymbol{c}_{p_\theta}(k) + \boldsymbol{c}_{p_\psi}(k)\right) \circ \boldsymbol{s} \circ \boldsymbol{n}_x \\
\boldsymbol{F}_{x_1}(k) &= -\left(\boldsymbol{c}_{p_{\dot{x}}}(k) + \boldsymbol{c}_{p_{\dot{y}}}(k) + \boldsymbol{c}_{p_{\dot{z}}}(k) + \boldsymbol{c}_{p_{\dot{\phi}}}(k) + \boldsymbol{c}_{p_{\dot{\theta}}}(k) + \boldsymbol{c}_{p_{\dot{\psi}}}(k)\right) \circ \boldsymbol{s} \circ \boldsymbol{n}_x \\
\boldsymbol{F}_{x_2}(k) &= -\left(\boldsymbol{c}_{p_{\ddot{x}}}(k) + \boldsymbol{c}_{p_{\ddot{y}}}(k) + \boldsymbol{c}_{p_{\ddot{z}}}(k)\right) \circ \boldsymbol{s} \circ \boldsymbol{n}_x \\
\boldsymbol{F}_{y_0}(k) &= -\left(\boldsymbol{c}_{p_\phi}(k) + \boldsymbol{c}_{p_\theta}(k) + \boldsymbol{c}_{p_\psi}(k)\right) \circ \boldsymbol{s} \circ \boldsymbol{n}_y \\
\boldsymbol{F}_{y_1}(k) &= -\left(\boldsymbol{c}_{p_{\dot{x}}}(k) + \boldsymbol{c}_{p_{\dot{y}}}(k) + \boldsymbol{c}_{p_{\dot{z}}}(k) + \boldsymbol{c}_{p_{\dot{\phi}}}(k) + \boldsymbol{c}_{p_{\dot{\theta}}}(k) + \boldsymbol{c}_{p_{\dot{\psi}}}(k)\right) \circ \boldsymbol{s} \circ \boldsymbol{n}_y \\
\boldsymbol{F}_{y_2}(k) &= -\left(\boldsymbol{c}_{p_{\ddot{x}}}(k) + \boldsymbol{c}_{p_{\ddot{y}}}(k) + \boldsymbol{c}_{p_{\ddot{z}}}(k)\right) \circ \boldsymbol{s} \circ \boldsymbol{n}_y \\
\boldsymbol{F}_{z_0}(k) &= -\left(\boldsymbol{c}_{p_\phi}(k) + \boldsymbol{c}_{p_\theta}(k) + \boldsymbol{c}_{p_\psi}(k)\right) \circ \boldsymbol{s} \circ \boldsymbol{n}_z \\
\boldsymbol{F}_{z_1}(k) &= -\left(\boldsymbol{c}_{p_{\dot{x}}}(k) + \boldsymbol{c}_{p_{\dot{y}}}(k) + \boldsymbol{c}_{p_{\dot{z}}}(k) + \boldsymbol{c}_{p_{\dot{\phi}}}(k) + \boldsymbol{c}_{p_{\dot{\theta}}}(k) + \boldsymbol{c}_{p_{\dot{\psi}}}(k)\right) \circ \boldsymbol{s} \circ \boldsymbol{n}_z \\
\boldsymbol{F}_{z_2}(k) &= -\left(\boldsymbol{c}_{p_{\ddot{x}}}(k) + \boldsymbol{c}_{p_{\ddot{y}}}(k) + \boldsymbol{c}_{p_{\ddot{z}}}(k)\right) \circ \boldsymbol{s} \circ \boldsymbol{n}_z
\end{aligned} \tag{30}
$$

Equation 28 is solved using determinant iteration for the system eigenvalues at each selected value of the free stream airspeed. The objective is to solve

$$
\begin{aligned}
\Bigg| \left(\frac{2Q_\infty^2}{c_0^2}\boldsymbol{A} - \frac{1}{2}\rho_\infty Q_\infty^2\boldsymbol{Q}_2(k)\right)p^2 \quad &+ \quad \left(\frac{2Q_\infty}{c_0}\boldsymbol{A} - \frac{1}{2}\rho_\infty Q_\infty^2\boldsymbol{Q}_1(k)\right)p \\
&+ \left(\boldsymbol{E} - \frac{1}{2}\rho_\infty Q_\infty^2\boldsymbol{Q}_0(k)\right)\Bigg| = 0
\end{aligned} \tag{31}
$$

for the reduced eigenvalue $p = g + \mathrm{i}k$, where g is the reduced damping and k the reduced frequency. Both the real and imaginary parts of the determinant must be equal to zero, so that there are two equations with two unknowns, g and k. The problem is nonlinear so it must be solved iteratively, starting from an initial guess $g = 0$ and $k = \omega_{n_i} c_0 / 2Q_\infty$, where ω_{n_i} is the ith wind-off natural frequency of the system. Once p_i has been evaluated for one airspeed, it is used as the initial guess for the next airspeed and so on, until p_i is available at all Q_∞ values of interest. Then, the entire process is repeated for all the other wind-off natural frequencies. Finally, the ith system eigenvalue at Q_∞ is calculated from

$$\lambda_i(Q_\infty) = \frac{2Q_\infty}{c_0} p_i(Q_\infty)$$

for $i = 1, \ldots, K$. The fact that k varies during the determinant iteration procedure means that $\boldsymbol{Q}_0(k)$, $\boldsymbol{Q}_1(k)$ and $\boldsymbol{Q}_2(k)$ must be calculated at each current value of k. In order to reduce the cost of the flutter solution, $\boldsymbol{Q}_0(k)$, $\boldsymbol{Q}_1(k)$ and $\boldsymbol{Q}_2(k)$ are calculated at a number of pre-selected k values and then interpolated as needed.

The system is stable as long as all $\lambda_i(Q_\infty)$ have negative real parts. This condition is expressed in `SDPMflut` in terms of the damping ratio

$$\zeta_i(Q_\infty) = -\frac{\Re(\lambda_i(Q_\infty))}{|\lambda_i(Q_\infty)|}$$

so that $\zeta_i(Q_\infty) > 0$ for stability. If any $\zeta_i(Q_\infty)$ becomes negative inside the airspeed range of interest, `SDPMflut` repeats the determinant iteration procedure in the neighbourhood of the sign change in order to pinpoint accurately the flutter airspeed Q_F at which $\zeta_i(Q_F) = 0$. Note that, since $g = 0$ at the flutter speed, then the flutter eigenvalue becomes $p = \mathrm{i}k_F$, k_F being the flutter reduced frequency; consequently, this second round of determinant iterations solves the problem

$$\left| \left(\frac{2Q_F^2}{c_0^2} \boldsymbol{A} - \frac{1}{2}\rho_\infty Q_F^2 \boldsymbol{Q}_2(k_F) \right) (\mathrm{i}k_F)^2 + \left(\frac{2Q_F}{c_0} \boldsymbol{A} - \frac{1}{2}\rho_\infty Q_F^2 \boldsymbol{Q}_1(k_F) \right) \mathrm{i}k_F \right.$$
$$\left. + \left(\boldsymbol{E} - \frac{1}{2}\rho_\infty Q_F^2 \boldsymbol{Q}_0(k_F) \right) \right| = 0 \qquad (32)$$

for the two unknowns Q_F and k_F. Then, the dimensional flutter frequency is obtained from

$$\omega_F = \frac{2Q_F}{c_0} k_F$$

3 Doublet Lattice Method

The Doublet Lattice Method solves equation 1 and then calculates the pressure using the linearized version of equation 2

$$c_p = -\frac{2\phi_x}{Q_\infty} - \frac{2}{Q_\infty^2}\phi_t \qquad (33)$$

The wing is modelled as a flat surface that is parallel to the x axis in the chordwise direction. Consequently, the DLM does not model wing thickness, camber or twist, although its nonplanar version can model dihedral. By integrating equation 33 over x and then differentiating it twice over z, the normalwash, $w(\mathbf{x},\omega)$, at any point on the wing $\mathbf{x}$ is given as a function of the pressure jump across the surface at $\mathbf{x}_s$, $\Delta p(\mathbf{x}_s,\omega)$, such that

$$w(\mathbf{x},\omega) = -\frac{1}{4\pi\rho_\infty U_\infty}\int_S \Delta p(\mathbf{x}_s,\omega)K(x_0,y_0,z_0,\omega)\mathrm{d}x_s\mathrm{d}y_s \tag{34}$$

where the Kernel function is defined as

$$K(x_0,y_0,z_0,\omega) = \frac{\partial^2}{\partial z^2}\left(\int_{-\infty}^{x_0}\frac{\mathrm{e}^{\frac{\mathrm{i}\omega}{U_\infty\beta^2}(\lambda - M_\infty r_\beta(\lambda,y_0,z_0))}}{r_\beta(\lambda,y_0,z_0)}\mathrm{d}\lambda\right)\mathrm{e}^{-\mathrm{i}\omega x_0/U_\infty} \tag{35}$$

with

$$x_0 = x - x_s,\;\; y_0 = y - y_s,\;\; z_0 = z - z_s$$

and

$$r_\beta(\lambda,y_0,z_0) = \sqrt{\lambda^2+\beta^2 y_0^2+\beta^2 z_0^2}$$

λ being a variable of integration over x. The normalwash is calculated from the unsteady boundary condition on the surface, such that equation 34 becomes a single equation with a single unknown, the pressure jump distribution across the surface $\Delta p(\mathbf{x}_s,\omega)$. However, it is an integral equation that cannot be evaluated analytically over general wing planforms.

3.1 The Doublet Lattice approximation

The Doublet Lattice Method solves equation 34 by applying two sets of approximations:

- The Kernel integral of equation 35 is evaluated by means of an exponential series approximation of the integrand. This evaluation is detailed in appendix B.1. The end result is the expression of the Kernel function in the form

$$K(x_0,y_0,z_0,\omega) = \mathrm{e}^{-\mathrm{i}\omega x_0/U_\infty}\frac{K_1T_1+K_2T_2}{r^2} \tag{36}$$

  where values for K_1, T_1, K_2, T_2 and r are given in appendix B.1.

- The wing's surface is discretized into quadrilateral panels, a doublet line is placed on the quarter chord of each panel and the normalwash is evaluated at a control point, lying at the midspan 3/4 chord point of the panel. The Kernel function is then integrated by means of a quadratic or quartic approximation.

The panel discretization results in N trapezoidal panels on which $\Delta p(\mathbf{x}_s,\omega)$ is assumed to be constant. Figure 1 demonstrates the DLM discretization scheme; the panels are aligned with the x direction and are of trapezoidal shape. Instead of a continuous surface doublet distribution, each panel features a doublet line with constant strength along its quarter-chord line. In replacing the surface integral in equation 34 by a line integral, one dimension is discarded and the equation is no longer dimensionally consistent. The usual approach is to integrate over y_s and to pre-multiply the integral by the constant $\Delta\bar{x}_J$, the mean chordwise length of the Jth panel. Equation 34 becomes

$$w_I(\omega) = -\frac{1}{4\pi\rho_\infty U_\infty}\sum_{J=1}^{N}\Delta p_J(\omega)\Delta\bar{x}_J\int_{S_J}\mathrm{e}^{-\mathrm{i}\omega x_0/U_\infty}\left(\frac{K_1T_1}{r^2}+\frac{K_2T_2^*}{r^4}\right)\mathrm{d}y_s \tag{37}$$

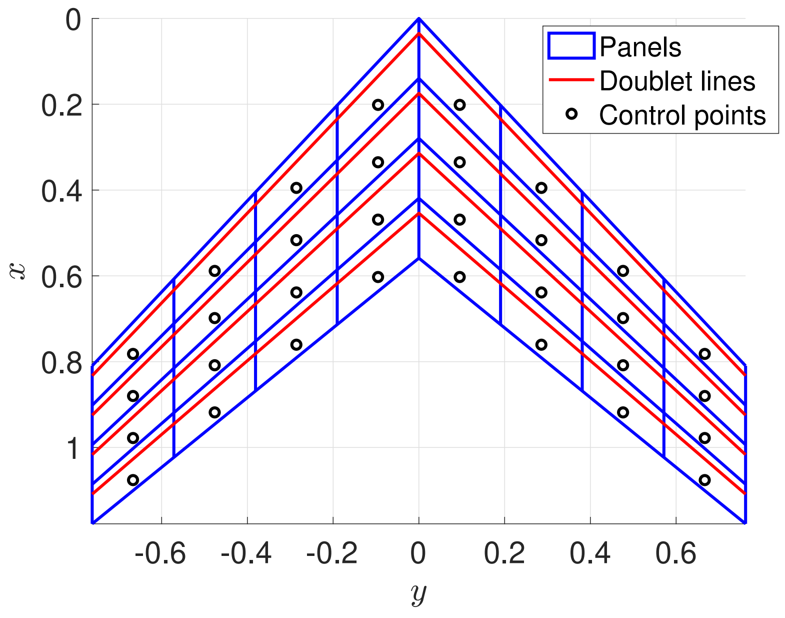


Figure 1: Panel grid used by the DLM

for $I = 1, \ldots, N$, where $T_2^* = r^2 T_2$, $w_I(\omega)$ is the value of the upwash at the control point of the Ith panel, which lies on the 3/4 chord midspan position, $\Delta p_J(\omega)$ is the pressure jump across the Jth panel and S_J is the surface area of the Jth panel. From this point onwards, subscript I will denote the influenced point and subscript J the influencing panel. It is customary to separate the integral in equation 37 into planar and nonplanar contributions. The equation is written in the form

$$w_I(\omega) = -\frac{1}{4\pi\rho_\infty U_\infty} \sum_{J=1}^{N} \Delta p_J(\omega) \left(D_{1_{I,J}}(\omega) + D_{2_{I,J}}(\omega) \right) \tag{38}$$

where

$$D_{1_{I,J}}(\omega) = \Delta\bar{x}_J \int_{S_J} \mathrm{e}^{-\mathrm{i}\omega x_0/U_\infty} \frac{K_1 T_1}{r^2} \mathrm{d}y_s \tag{39}$$

$$D_{2_{I,J}}(\omega) = \Delta\bar{x}_J \int_{S_J} \mathrm{e}^{-\mathrm{i}\omega x_0/U_\infty} \frac{K_2 T_2^*}{r^4} \mathrm{d}y_s \tag{40}$$

Note that, from this point onwards, the influenced point will always be denoted by subscript I and the influencing panel by subscript J. The $D_{1_{I,J}}(\omega)$ integral is known as the planar contribution and $D_{2_{I,J}}(\omega)$ as the nonplanar contribution, usually referred to as the planar and nonplanar upwash factors. The evaluation of the upwash factors is detailed in appendices B.2 and B.3. It is important to note that the integral of the Kernel function over the doublet line currently relies on a quadratic approximation in SDPMflut. A quartic approximation [13] will be implemented at a later stage.

3.2 The DLM boundary condition

It is assumed that the surface is immersed in a steady free stream $\mathbf{Q}_\infty = (U_\infty,\ V_\infty,\ W_\infty)$ and undergoes small amplitude oscillations in the direction of the structural mode shapes, $\boldsymbol{\Phi}_x$, $\boldsymbol{\Phi}_y$, $\boldsymbol{\Phi}_z$, $\boldsymbol{\Phi}_\phi$, $\boldsymbol{\Phi}_\theta$, $\boldsymbol{\Phi}_\psi$ but the aerodynamic modelling treats the surface as quasi-fixed. Under these assumptions, [4] show that the total relative flow velocity due to flexible motion of a surface can be written as

$$\begin{aligned}
\boldsymbol{u}_{\text{rel}}(t) &= U_\infty + V_\infty \boldsymbol{\Phi}_\psi \boldsymbol{q}(t) - W_\infty \boldsymbol{\Phi}_\theta \boldsymbol{q}(t) - \boldsymbol{\Phi}_x \dot{\boldsymbol{q}}(t) \\
\boldsymbol{v}_{\text{rel}}(t) &= -U_\infty \boldsymbol{\Phi}_\psi \boldsymbol{q}(t) + V_\infty + W_\infty \boldsymbol{\Phi}_\phi \boldsymbol{q}(t) - \boldsymbol{\Phi}_y \dot{\boldsymbol{q}}(t) \\
\boldsymbol{w}_{\text{rel}}(t) &= U_\infty \boldsymbol{\Phi}_\theta \boldsymbol{q}(t) - V_\infty \boldsymbol{\Phi}_\phi \boldsymbol{q}(t) + W_\infty - \boldsymbol{\Phi}_z \dot{\boldsymbol{q}}(t)
\end{aligned} \tag{41}$$

where $\boldsymbol{q}(t)$ is the $M \times 1$ vector of generalized coordinates, M being the number of modes. Taking the Fourier transform leads to

$$\begin{aligned}
\boldsymbol{u}_{\text{rel}}(\omega) &= U_\infty \delta_{\omega,0} + V_\infty \boldsymbol{\Phi}_\psi \boldsymbol{q}(\omega) - W_\infty \boldsymbol{\Phi}_\theta \boldsymbol{q}(\omega) - \mathrm{i}\omega \boldsymbol{\Phi}_x \boldsymbol{q}(\omega) \\
\boldsymbol{v}_{\text{rel}}(\omega) &= -U_\infty \boldsymbol{\Phi}_\psi \boldsymbol{q}(\omega) + V_\infty \delta_{\omega,0} + W_\infty \boldsymbol{\Phi}_\phi \boldsymbol{q}(\omega) - \boldsymbol{\Phi}_y \boldsymbol{q}(\omega) \\
\boldsymbol{w}_{\text{rel}}(\omega) &= U_\infty \boldsymbol{\Phi}_\theta \boldsymbol{q}(\omega) - V_\infty \boldsymbol{\Phi}_\phi \boldsymbol{q}(\omega) + W_\infty \delta_{\omega,0} - \boldsymbol{\Phi}_z \boldsymbol{q}(\omega)
\end{aligned} \tag{42}$$

where $\delta_{\omega,0}$ is the Kronecker Delta function. Dividing throughout by $Q_\infty = ||\mathbf{Q}_\infty||$ leads to the normalized relative velocities

$$\begin{aligned}
\bar{\boldsymbol{u}}_{\text{rel}}(k) &= \bar{U}_\infty \delta_{k,0} + \bar{V}_\infty \boldsymbol{\Phi}_\psi \boldsymbol{q}(k) - \bar{W}_\infty \boldsymbol{\Phi}_\theta \boldsymbol{q}(k) - \frac{\mathrm{i}k}{b_{\text{char}}} \boldsymbol{\Phi}_x \boldsymbol{q}(k) \\
\bar{\boldsymbol{v}}_{\text{rel}}(k) &= -\bar{U}_\infty \boldsymbol{\Phi}_\psi \boldsymbol{q}(k) + \bar{V}_\infty \delta_{k,0} + \bar{W}_\infty \boldsymbol{\Phi}_\phi \boldsymbol{q}(k) - \frac{\mathrm{i}k}{b_{\text{char}}} \boldsymbol{\Phi}_y \boldsymbol{q}(k) \\
\bar{\boldsymbol{w}}_{\text{rel}}(k) &= \bar{U}_\infty \boldsymbol{\Phi}_\theta \boldsymbol{q}(k) - \bar{V}_\infty \boldsymbol{\Phi}_\phi \boldsymbol{q}(k) + \bar{W}_\infty \delta_{k,0} - \frac{\mathrm{i}k}{b_{\text{char}}} \boldsymbol{\Phi}_z \boldsymbol{q}(k)
\end{aligned} \tag{43}$$

where k is the reduced frequency and

$$U_\infty = Q_\infty \cos\alpha_0 \cos\beta_0,\ \ V_\infty = -Q_\infty \sin\beta_0,\ \ W_\infty = Q_\infty \sin\alpha_0 \cos\beta_0 \tag{44}$$

α_0 being the steady angle of attack and β_0 the steady sideslip angle. If it is assumed that $\alpha_0 \ll 1$, $\beta_0 \ll 1$, the reduced frequency in equation 43 becomes approximately equal to that of equation 165, i.e.

$$\frac{\omega b_{\text{char}}}{Q_\infty} \approx \frac{\omega b_{\text{char}}}{U_\infty}$$

The quantity $w_I(k)$ in equation 168 is the relative velocity normal to the surface, the normalwash. It can be calculated by taking the scalar product of the normalized relative velocity on the Ith panel by the unit normal vector to the panel, such that

$$\bar{w}_I(k) = \bar{u}_{\text{rel}_I}(k) n_{x_I} + \bar{v}_{\text{rel}_I}(k) n_{y_I} + \bar{w}_{\text{rel}_I}(k) n_{z_I}$$

The DLM models the surface as parallel to the x axis, such that $n_{x_I} = 0$. Consequently, equation 168 is written in matrix-vector notation as

$$-\bar{\boldsymbol{w}}(k) = -\bar{\boldsymbol{v}}_{\text{rel}}(k) \circ \boldsymbol{n}_y - \bar{\boldsymbol{u}}_{\text{rel}}(k) \circ \boldsymbol{n}_z = \boldsymbol{A}(k) \Delta \boldsymbol{c}_p(k) \tag{45}$$

where $\circ$ denotes the Hadamard product, $\boldsymbol{n}_y$, $\boldsymbol{n}_z$, $\Delta\boldsymbol{c}_p(k)$ are the $N \times 1$ vectors of the normal vector components and pressure jump values on the N panels and

$$\boldsymbol{A}(k) = \frac{1}{8\pi}\left(\boldsymbol{D}_1(k) + \boldsymbol{D}_2(k)\right)$$

is the total $N \times N$ aerodynamic influence coefficient matrix. The normalwash vector $\bar{\boldsymbol{w}}_I(k)$ is given by

$$\begin{aligned}\bar{\boldsymbol{w}}(0) &= \bar{V}_\infty \boldsymbol{n}_y + \bar{W}_\infty \boldsymbol{n}_z + \left[\left(-\bar{U}_\infty \boldsymbol{\Phi}_\psi + \bar{W}_\infty \boldsymbol{\Phi}_\phi\right) \circ \boldsymbol{n}_y \right.\\ &\quad \left. + \left(\bar{U}_\infty \boldsymbol{\Phi}_\theta - \bar{V}_\infty \boldsymbol{\Phi}_\phi\right) \circ \boldsymbol{n}_z\right] \boldsymbol{q}(0) \end{aligned} \tag{46}$$

for $k = 0$ and

$$\begin{aligned}\bar{\boldsymbol{w}}(k) &= \left[\left(-\bar{U}_\infty \boldsymbol{\Phi}_\psi + \bar{W}_\infty \boldsymbol{\Phi}_\phi - \frac{\mathrm{i}k}{b_{\text{char}}}\boldsymbol{\Phi}_y\right) \circ \boldsymbol{n}_y \right.\\ &\quad \left. + \left(\bar{U}_\infty \boldsymbol{\Phi}_\theta - \bar{V}_\infty \boldsymbol{\Phi}_\phi - \frac{\mathrm{i}k}{b_{\text{char}}}\boldsymbol{\Phi}_z\right) \circ \boldsymbol{n}_z\right] \boldsymbol{q}(k) \end{aligned} \tag{47}$$

for $k \neq 0$. Equation 45 can then be solved for $\Delta\boldsymbol{c}_p(k)$ as a function of $\boldsymbol{q}(k)$

$$\Delta\boldsymbol{c}_p(k) = -\boldsymbol{A}(k)^{-1}\bar{\boldsymbol{w}}(k)$$

The aerodynamic loads acting on the panels are given by

$$\begin{aligned}\boldsymbol{F}_x(k) &= \Delta\boldsymbol{c}_p(k) \circ \boldsymbol{s} \circ \boldsymbol{n}_x\\ \boldsymbol{F}_y(k) &= \Delta\boldsymbol{c}_p(k) \circ \boldsymbol{s} \circ \boldsymbol{n}_y\\ \boldsymbol{F}_z(k) &= \Delta\boldsymbol{c}_p(k) \circ \boldsymbol{s} \circ \boldsymbol{n}_z\end{aligned} \tag{48}$$

where $\boldsymbol{s}$ is the $N \times 1$ vector of the areas of the panels. Finally, the generalized aerodynamic force is given by

$$\boldsymbol{Q}(k) = \boldsymbol{\Phi}_x^T \boldsymbol{F}_x(k) + \boldsymbol{\Phi}_y^T \boldsymbol{F}_y(k) + \boldsymbol{\Phi}_z^T \boldsymbol{F}_z(k)$$

The aeroelastic equation becomes

$$\boldsymbol{A}\ddot{\mathbf{q}} + \boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{E}\mathbf{q} = \frac{1}{2}\rho_\infty Q_\infty^2 \boldsymbol{Q}(k)\mathbf{q} \tag{49}$$

and can be solved using several flutter techniques, such as the p-k method [14] or the g method [15].

Noting that equation 47 includes terms that depend on rotations ϕ, θ, ψ and translations x, y, it is possible to defined the pressure derivatives

$$\begin{aligned}\Delta\boldsymbol{c}_{p_\phi}(k) &= \boldsymbol{A}(k)^{-1}\left(-\bar{W}_\infty \boldsymbol{\Phi}_\phi \circ \boldsymbol{n}_y + \bar{V}_\infty \boldsymbol{\Phi}_\phi \circ \boldsymbol{n}_z\right)\\ \Delta\boldsymbol{c}_{p_\theta}(k) &= -\boldsymbol{A}(k)^{-1}\bar{U}_\infty \boldsymbol{\Phi}_\theta \circ \boldsymbol{n}_z\\ \Delta\boldsymbol{c}_{p_\psi}(k) &= \boldsymbol{A}(k)^{-1}\bar{U}_\infty \boldsymbol{\Phi}_\psi \circ \boldsymbol{n}_y\\ \Delta\boldsymbol{c}_{p_{\dot{y}}}(k) &= \frac{1}{b_{\text{char}}}\boldsymbol{A}(k)^{-1}\boldsymbol{\Phi}_y \circ \boldsymbol{n}_y\\ \Delta\boldsymbol{c}_{p_{\dot{z}}}(k) &= \frac{1}{b_{\text{char}}}\boldsymbol{A}(k)^{-1}\boldsymbol{\Phi}_z \circ \boldsymbol{n}_z\end{aligned}$$

so that

$$\Delta \boldsymbol{c}_p(k) = \left(\Delta \boldsymbol{c}_{p_0}(k) + \mathrm{i}k\Delta \boldsymbol{c}_{p_1}(k)\right) \boldsymbol{q}(k)$$

where

$$\begin{aligned}\Delta \boldsymbol{c}_{p_0}(k) &= \Delta \boldsymbol{c}_{p_\phi}(k) + \Delta \boldsymbol{c}_{p_\theta}(k) + \Delta \boldsymbol{c}_{p_\psi}(k) \\ \Delta \boldsymbol{c}_{p_1}(k) &= \Delta \boldsymbol{c}_{p_{\dot{y}}}(k) + \Delta \boldsymbol{c}_{p_{\dot{z}}}(k)\end{aligned}$$

4 Installing and running SDPMflut

`SDPMflut` is written in Python and C. To run it you need to have a recent Python distribution and a C compiler installed on your system. The installation steps are the following:

1. Unzip the `SDPMflut_v0.72.zip` file, place anywhere in your file space, then edit the line starting with `install_dir=` in
   - `AGARD445_6/flutter_SDPM_AGARD.py`
   - `NACARML5/unsteady_SDPM_delta.py`
   - `NACARML5/unsteady_SDPM_straight.py`
   - `NACARML5/unsteady_SDPM_swept.py`
   - `NASATMX72799/flutter_SDPM_NASATMX72799.py`
   - `NASATND344/unsteady_SDPM_NASATND344.py`
   - `NASATM84367/steady_SDPM_NASATM84367.py`
   - `PAPA/flutter_SDPM_NACA0012.py`
   - `PAPA/flutter_SDPM_BSCW.py`
   - `PAPA/flutter_SDPM_NACA64A010.py`
   - `T_tail/flutter_SDPM_Ttail.py`.
   - `Theodorsen/flutter_SDPM_Theodorsen.py`.

   and in the same file names with `DLM` instead of `SDPM` (the `NACARML5` test cases have no DLM solution yet). You need to give the absolute path to the `Common` directory, as installed on your system. Mac OS/Linux example:
   ```
   install_dir=r"/Users/Username/Documents/Python/SDPMflut_v0.6/Common/"
   ```
   Windows example:
   ```
   install_dir=r"C:\Users\Username\Documents\Python\SDPMflut\Common"
   ```

2. Compile the C codes found in directory `Common`. At your C compiler's terminal type:
   - `gcc -fPIC -shared -o sdpminfso.so sdpminfso.c`
   - `gcc -fPIC -shared -o sdpminf_unsteadyso.so sdpminf_unsteadyso.c`

- `gcc -fPIC -shared -o dlminf_nonplanarso.so dlminf_nonplanarso.c`—

assuming that you have GNU C installed. The same commands with `cc` instead of `gcc` may work.

If you do not have a Python distribution or IDE on your system you can install Anaconda and Spyder:

1. Download Anaconda from https://docs.anaconda.com/anaconda/install/.
2. Run the installer, selecting the default installation options.
3. On Windows, Anaconda installs its own version of the DOS command window, called Anaconda Prompt. Launch Anaconda Prompt.
4. On Mac OS/Linux launch the terminal.
5. Type `conda - - version` (no space between the dashes).
6. Instal Spyder from https://www.spyder-ide.org.
7. Launch Spyder and go to `Preferences->Python interpreter`
8. Click on `Use the following Python interpreter` and select
   `/opt/anaconda3/bin/python`
   or whatever version has been installed by Anaconda.
9. Click `Apply`, `OK` and then from the `Consoles` menu select `Restart kernel`
10. It is likely that Spyder will complain that the interpreter you selected does not have the right version of the Spyder Kernels.
11. Launch the Anaconda Prompt or terminal as Administrator. Type:
    `conda install spyder-kernels=3.0`
    or whatever version Spyder has asked for.
12. Quit and restart Spyder.

If you do not have a C compiler on your system, you can install one depending on your system architecture:

- On Mac OS install Xcode and the command line tools. You can also optionally instal GCC (GNU Compiler Collection).
- On Linux install GCC (GNU Compiler Collection).
- On Windows there are many options. For example, you can download MinGW from https://www.mingw-w64.org/downloads/. Extract the zip file to `C:\Prog` or wherever else you wish. Launch the Command Prompt and set the path:
  - Temporary option: type `set path=C:\Prog\mingw64\bin;%PATH%`
  - Permanent option: type `setx PATH ^%PATH^%;"C:\Prog\mingw64\bin"`

To run `SDPMflut`, Launch Spyder and open one of the test cases. For example, you can open `PAPA/flutter_SDPM_NACA0012.py`. Click on `Rune file` or type `F5`.

5 Structure of SDPMflut

`SDPMflut` does not have a user interface. For each test case there is a subdirectory inside the installation directory that contains one or more `.py` files and may also contain one or more `.mat` files. The file names of the files to run end with `.py` and start with

- `steady_SDPM_`: These files calculate only steady aerodynamic pressures and loads using the SDPM.
- `unsteady_SDPM_`: These files calculate both steady and unsteady aerodynamic pressures and loads using the SDPM.
- `flutter_SDPM_`: These files calculate steady and unsteady aerodynamic pressures and loads, as well as flutter solutions using the SDPM.
- `steady_DLM_`: These files calculate only steady aerodynamic pressures and loads using the DLM.
- `unsteady_DLM_`: These files calculate both steady and unsteady aerodynamic pressures and loads using the DLM.
- `flutter_DLM_`: These files calculate steady and unsteady aerodynamic pressures and loads, as well as flutter solutions using the DLM.

The current distribution contains the following test cases:

- `AGARD445_6`: Flutter solution for the weakened AGARD 445.6 wing described in [16].
- `NACARMA51G31`: Steady aerodynamic solution for the swept and tapered wing described in [17].
- `NACARML5`: Calculation of aerodynamic stability derivatives for a straight tapered wing, a swept tapered wing and a delta wing, described in [18, 19, 20, 21].
- `NASATMX72799`: Flutter solution for the flat plate swept wing with and without winglets described in [22].
- `NASATND344`: Steady and unsteady pressure calculation for the rectangular wing forced to oscillate in [23].
- `NASATM84367`: Steady pressure calculation for the swept wing described in [24].
- `PAPA`: Flutter solution for the rectangular NACA 0012, NACA 64A010 and BSCW wings tested by NASA on the Pitch and Plunge Apparatus (PAPA) described in [25, 26, 27].
- `T-tail`: Flutter solution for the rectangular Van Zyl T-tail described in [28, 29, 30].
- `Theodorsen`: Flutter solution for a 2D infinitely thin flat plate airfoil with pitch and plunge degrees of freedom at incompressible conditions and comparison to Theodorsen theory results [31].

The filename of each of the run files ends with the name of the test case. When the file is run, it calls functions in the `Common` directory; it may also call functions in the test case directory and load data from `.mat` files in the test case directory.

The structure of each run file is the following:

1. Load libraries, C functions and data types.
2. Define flight conditions.
3. Input geometries for all bodies in the flow.
4. Optional: Input structural models for all bodies.
5. For each of the selected flight conditions:
    - Calculate the steady aerodynamic pressures and loads at prescribed values of the Mach number, angle of attack and angle of sideslip.
    - Optional: For each of the selected reduced frequencies calculate the unsteady aerodynamic pressures derivatives with respect to rigid-body or flexible motion coordinates.
    - Optional: Carry out flutter analysis to calculate the flutter speed and frequency.
6. Plot results.

5.1 Geometry definition

Only wing geometry definitions are included in the current distribution; fuselage definitions will be included in a future release. Wings are defined using trapezoidal sections, defined in the `tp_trap` data type. Each element of `tp_trap` is a trapezoidal section with the following parameters:

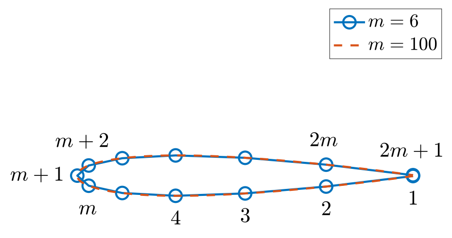
- `rootchord`: The chord length of the root of the current trapezoidal section.
- `xledist`: The chordwise distance of the root of the current trapezoidal section to the tip of the previous trapezoidal section.
- `span`: The span of the current trapezoidal section.
- `taper`: The taper ratio of the current trapezoidal section, i.e. the ratio of the tip chord to the root chord of the trapezoidal section.
- `sweepLE`: The sweep angle at the leading edge of the current trapezoidal section.
- `roottwist`: The twist angle at the root of the current trapezoidal section.
- `tiptwist`: The twist angle at the tip of the current trapezoidal section.
- `twistcent`: The non-dimensional chordwise position of the axis around which the wing is twisted, taking values between 0 and 1.

- `dihedral`: The dihedral angle of the current trapezoidal section.
- `rootairfoil`: The name of the root airfoil of the current trapezoidal section. This name must be a function defined in `Common/airfoils.py`. If your airfoil is not already defined (most likely), you need to write a function for it in `Common/airfoils.py`.
- `rootairfoilparams`: The parameters of the root airfoil. These depend on the airfoil type and its definition; they may be thickness, camber, NACA number etc. For NACA four- and five-digit airfoils the parameter `teclosed` determines the thickness of the trailing edge. If `teclosed=0`, the original thickness equation is used and the trailing edge thickness is finite. If `teclosed=1`, a modified thickness equation is used and the trailing edge thickness is zero.
- `tipairfoil`: The name of the tip airfoil of the current trapezoidal section. The root and tip airfoils are interpolated linearly to obtain the airfoil shape in between.
- `tipairfoilparams`: The parameters of the tip airfoil.

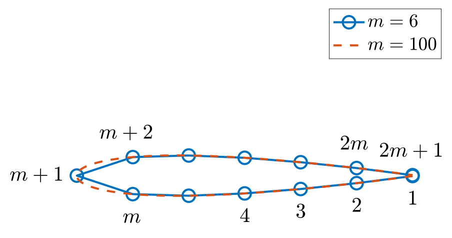
There can be any number of trapezoidal sections; new sections are needed when any of the parameters defined in `tp_trap` change. The root and tip airfoil and their parameters can be identical. The SDPM grid for the complete wing is created using the parameters for all trapezoidal sections by calling function `SDPMgeometry_trap_fun` in package `SDPMgeometry.py` with the following inputs:

- `body`: Structured array of type `tp_body` (see later) holding the SDPM grid information for all bodies present in the flow.
- `ibody`: The index of `body` for which the SDPM grid is to be calculated.
- `m`: Number of chordwise panels on the upper surface of the wing. There will be another `m` panels on the lower side for a total of `2*m` panels. The value of `m` is constant for all trapezoidal sections.
- `mw`: Number of chordwise panels in the wake. This is calculated as `mw=m*nchords`, where `nchords` is the length of the wake in root chord lengths. Recommended value: `nchords=10`.
- `nhalf`: Number of spanwise panels on the half-wing (semispan wing). If a full-span wing is requested (`mirroredwing=2`, see later), there will be another `nhalf` on the other half for a total of `2*nhalf` panels.
- `mirroredwing`: How to arrange the wing. Each wing is initially created as a right half-wing (semispan wing). If `mirroredwing=-1` it is changed to a left half-wing. If `mirroredwing=1` it remains a right half-wing. If `mirroredwing=2` a full-span wing is created by mirroring the right half-wing to the left.
- `linchord`: The distribution of chordwise panels. if `linchord=0` the distribution will be such that panel density is highest around the leading edge and lowest around the trailing edge. If `linchord=1` the chordwise distribution will have constant density.

- `linspan`: The distribution of spanwise panels. if `linspan=0` the distribution will be such that panel density is highest near the wingtip and root. If `linspan=1` the spanwise distribution will have constant density.
- `trap`: Structured array of type `tp_trap` containing the geometrical information for all trapezoidal sections for the current wing.
- `name`: The name of the wing. This is not important for the calculation, it only helps identify the wing when more than one wings are defined.
- `dir_tau`: Direction in which the unit tangent vector for this wing has a zero component. `dir_tau=1` denotes the x direction and is suitable for fuselages. `dir_tau=2` denotes the y direction and is suitable for horizontal wings. `dir_tau=3` denotes the z direction and is suitable for vertical wings and fins.
- `rollpitchyaw`: A three-element vector containing angles by which to roll, pitch and yaw the complete wing. To define a fin, set:
  `rollpitchyaw=np.array([0, 0, 90])*np.pi/180`
  Do not use `rollpitchyaw` to impose a dihedral angle, use the `dihedral` parameter of the `tp_trap` data type instead.
- `rollpitchyaw_cent`: The point around which the rotations defined in `rollpitchyaw` are performed.
- `lexyz`: A three-element array containing the coordinates of the root leading edge.
- `nmin`: The minimum number of spanwise panels in each trapezoidal section. Recommended value: `nmin=3`.



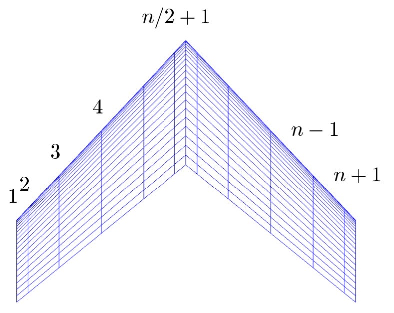
(a) linchord=0



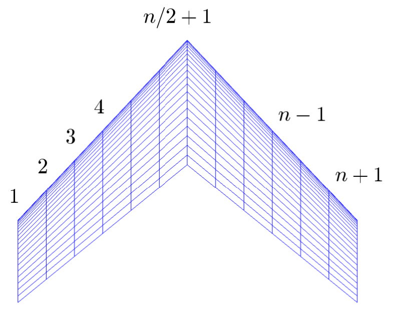
(b) linchord=1

Figure 2: Chordwise panel numbering and distributions for the SDPM

Figure 2 illustrates the chordwise panelling scheme of wings used by the SDPM solver in `SDPMflut`. There are m panels on the lower side and m panels on the upper side of the wing section, for a total of $2m$ panels and $2m + 1$ panel vertices. The panel vertex numbering starts with 1 at the lower trailing edge point and proceeds upstream, so that the leading edge is vertex $m+1$. The numbering then proceeds downstream on the upper surface, so that the upper trailing edge is vertex $2m+1$. Figure 2(a) plots the non-uniform chordwise panel distribution (`linchord=0`), whereby the paneling is denser around the leading edge. Figure 2(b) plots the uniform chordwise panel distribution (`linchord=1`), whereby the paneling is constant all around the wing section. It can be seen that the non-uniform distribution matches more closely the exact shape of the airfoil around the leading edge. The DLM solver only uses the uniform chordwise panel distribution on a single surface, so that there are m chordwise panels numbered from leading edge to trailing edge.



(a) linspan=0



(b) linspan=1

Figure 3: Spanwise panel numbering and distributions for the SDPM

Figure 3 illustrates the spanwise panelling scheme of wings used by the SDPM solver in `SDPMflut`. There are $n/2$ panels on the left side and $n/2$ panels on the right side of the wing, for a total of n panels and $n + 1$ panel vertices. The panel vertex numbering starts with 1 at the left wingtip and proceeds to the right, so that the centreline is vertex $n/2+1$ and the right wingtip is vertex $n + 1$. Figure 3(a) plots the non-uniform spanwise panel distribution (`linspan=0`), whereby the paneling is denser around the wingtips and centreline. Figure 3(b) plots the uniform spanwise panel distribution (`linspan=1`), whereby the paneling is constant all along the span. The DLM solver uses only the uniform spanwise panel distribution.

Figure 4 draws a right half-wing created by `SDPMflut` using three trapezoidal sections, for use with the SDPM solver. The inboard section has a high taper ratio, the middle section a lower taper ratio and the outer section a high dihedral angle, a shorter root chord and a different airfoil. The figure also demonstrates the coordinate system used by `SDPMflut`: the x axis points downstream, the y axis towards the right wingtip and the z axis upwards. Note that this axis system is different to the usual body-fixed axes used in flight dynamics, whereby the x axis points upstream and the z axis downwards. Finally, the figure also plots a section of the wake model; the wake panels are attached to the

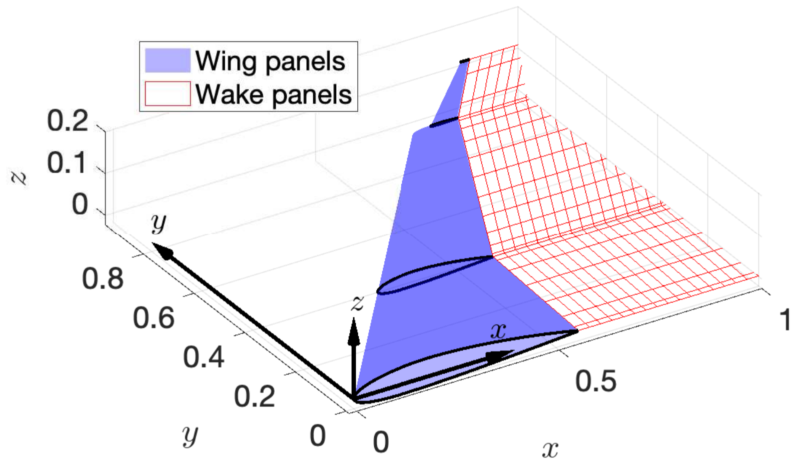


Figure 4: Coordinate system used by SDPMflut

lower trailing edge segments of their corresponding wing panels and extend downstream in the x direction. The chordwise spacing of the panels is set to c_0/m, where c_0 is the wing's root chord. The DLM solvers uses the same coordinate system but features only one wing surface and no wake model.

The geometries of all the bodies of a particular test case are stored in an array of type `tp_body`. Each element of this array type contains the following data:

- `m`: Number of chordwise panels on the upper surface of the wing.
- `n`: Number of spanwise panels on the upper surface of the wing. This number is set by function `SDPMgeometry_trap_fun`, depending on the values of `nhalf`, `nmin` and `mirroredwing`.
- `mw`: Number of chordwise panels in the wake.
- `Xp0`, `Yp0`, `Zp0`: $(2m+1) \times (n+1)$ matrices containing the x, y, z coordinates of the wing panel vertices in cartesian coordinates.
- `Xc0`, `Yc0`, `Zc0`: $2m \times n$ matrices containing the x, y, z coordinates of the wing panel control points (centroids) in cartesian coordinates.
- `Xc0all`, `Yc0all`, `Zc0all`: $2mn \times 1$ vectors containing the x, y, z coordinates of the wing panel control points (centroids) in cartesian coordinates.
- `Xw0`, `Yw0`, `Zw0`: $(m_w+1) \times (n+1)$ matrices containing the x, y, z coordinates of the wake panel vertices in cartesian coordinates.

- `nx0`, `ny0`, `nz0`: $2m \times n$ matrices containing the x, y, z components of unit vectors normal to the wing panels and pointing outwards in cartesian coordinates.
- `nx0all`, `ny0all`, `nz0all`: $2mn \times 1$ vectors containing the x, y, z components of unit vectors normal to the wing panels and pointing outwards in cartesian coordinates.
- `s0`: $2m \times n$ matrix containing the surface areas of the wing panels in cartesian coordinates.
- `s0all`: $2mn \times 1$ vector containing the surface areas of the wing panels in cartesian coordinates.
- `c0`: A reference chord length, currently it is the root chord.
- `b`: The total span, calculated from the trapezoidal section information stored in array `trap`.
- `S`: The planform area, calculated from the trapezoidal section information stored in array `trap`.
- `AR`: Aspect ratio of the full-span wing.
- `Xp`, `Yp`, `Zp`: $(2m+1) \times (n+1)$ matrices containing the ξ, η, ζ coordinates of the wing panel vertices in Prandtl-Glauert coordinates.
- `Xc`, `Yc`, `Zc`: $2m \times n$ matrices containing the ξ, η, ζ coordinates of the wing panel control points (centroids) in Prandtl-Glauert coordinates.
- `Xcall`, `Ycall`, `Zcall`: $2mn \times 1$ vectors containing the ξ, η, ζ coordinates of the wing panel control points (centroids) in Prandtl-Glauert coordinates.
- `Xw`, `Yw`, `Zw`: $(m_w+1) \times (n+1)$ matrices containing the ξ, η, ζ coordinates of the wake panel vertices in Prandtl-Glauert coordinates.
- `nx`, `ny`, `nz`: $2m \times n$ matrices containing the ξ, η, ζ components of unit vectors normal to the wing panels and pointing outwards in Prandtl-Glauert coordinates.
- `nxall`, `nyall`, `nzall`: $2mn \times 1$ vectors containing the ξ, η, ζ components of unit vectors normal to the wing panels and pointing outwards in Prandtl-Glauert coordinates.
- `s`: $2m \times n$ matrix containing the surface areas of the wing panels in Prandtl-Glauert coordinates.
- `sall`: $2mn \times 1$ vector containing the surface areas of the wing panels in Prandtl-Glauert coordinates.
- `tauxx`, `tauxy`, `tauxz`: $2m \times n$ matrices containing the ξ, η, ζ components of unit vectors tangent to the wing panels in Prandtl-Glauert coordinates. These vectors are aligned according to the value of `dir_tau`.

- `tauyx`, `tauyy`, `tauyz`: $2m \times n$ matrices containing the ξ, η, ζ components of another set of unit vectors tangent to the wing panels in Prandtl-Glauert coordinates. These vectors are orthogonal to vectors `nx`, `ny`, `nz` and `tauxx`, `tauxy`, `tauxz`.

- `tmx`, `tmy`, `tmz`: $2m \times n$ matrices containing the ξ, η, ζ components of unit vectors tangent to the wing panels and pointing in the chordwise direction in Prandtl-Glauert coordinates.

- `tnx`, `tny`, `tnz`: $2m \times n$ matrices containing the ξ, η, ζ components of unit vectors tangent to the wing panels and pointing in the spanwise direction in Prandtl-Glauert coordinates.

- `sm`, `sn`: $2m \times n$ matrices containing the mean chordwise and spanwise lengths, respectively, of the wing panels in Prandtl-Glauert coordinates.

- `Xcw`, `Ycw`, `Zcw`: $m_w \times n$ matrices containing the ξ, η, ζ coordinates of the wake panel control points (centroids) in Prandtl-Glauert coordinates.

- `nxw`, `nyw`, `nzw`: $m_w \times n$ matrices containing the ξ, η, ζ components of unit vectors normal to the wake panels and pointing upwards in Prandtl-Glauert coordinates.

- `tauxxw`, `tauxyw`, `tauxzw`: $m_w \times n$ matrices containing the ξ, η, ζ components of unit vectors tangent to the wake panels in Prandtl-Glauert coordinates. These vectors are aligned according to the value of `dir_tau`.

- `tauyxw`, `tauyyw`, `tauyzw`: $m_w \times n$ matrices containing the ξ, η, ζ components of another set of unit vectors tangent to the wake panels in Prandtl-Glauert coordinates. These vectors are orthogonal to vectors `nxw`, `nyw`, `nzw` and `tauxxw`, `tauxyw`, `tauxzw`.

- `cp0`: $2m \times n$ matrix containing the steady pressure coefficients on the wing panels.

- `Fx0`, `Fy0`, `Fz0`: $2m \times n$ matrices containing the x, y, z components of the aerodynamic forces acting on the wing panels in Newtons per Pascal. In order to obtain the aerodynamic forces in Newtons they must be multiplied by the free stream dynamic pressure. In order to obtain the forces in force coefficient form they must be divided by the reference area.

- `Mx0`, `My0`, `Mz0`: $2m \times n$ matrices containing the aerodynamic moments acting on the wing panels around the x, y, z axes in Newton meters per Pascal. In order to obtain the aerodynamic moments in Newtons they must be multiplied by the free stream dynamic pressure and a reference length. In order to obtain the moments in moment coefficient form they must be divided by the reference area.

- `mu0`: $2m \times n$ matrix containing the steady doublet strengths on the wing panels.

- `muw0`: $2m \times n$ matrix containing the steady doublet strengths on the wake panels.

- `Phi_xall`, `Phi_yall`, `Phi_zall`: $2mn \times K$ vectors containing the modal translations in the x, y, z directions at the wing panel control points.

- `Phi_phiall`, `Phi_thetaall`, `Phi_psiall`: $2mn \times K$ vectors containing the modal rotations around the x, y, z axes at the wing panel control points.

Function `SDPMgeometry_trap_fun` calculates all quantities in cartesian coordinates. All quantities in Prandtl-Glauert coordinates are calculated by function `PGtransform` in package `SDPMcalcs.py`. The steady aerodynamic calculations are carried out in the run file. The modal translations and rotations are calculated by function `SDPMmodeinterp` in package `FEmodes.py`.

Wing geometries for the DLM are created by function `DLMgeometry_trap_fun`. It is very similar to `SDPMgeometry_trap_fun` but there are differences in the quantities it calculates. As a consequence the DLM `tp_body` array contains different fields, such as the doublet line vertices and midpoint, the mean spanwise and chordwise lengths of the panels, the dihedral angles γ_s, and does not contain any quantities in Prandtl-Glauert coordinates, unit tangent vectors, or wake-related quantities. Furthermore, DLM distances and lengths are either physical or normalized by a characteristic length `bchar` set by the user. The modal translations and rotations are calculated by function `DLMmodeinterp` in package `FEmodes.py`.

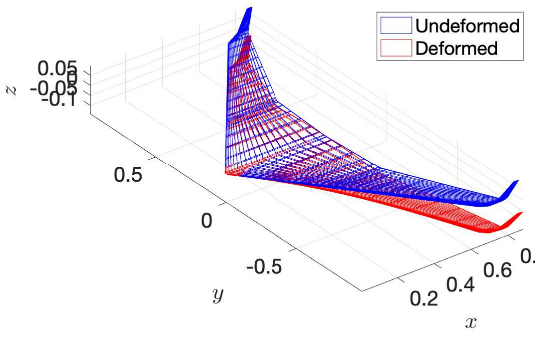
6 Structural model input

The structural model is composed of the $K \times K$ structural modal matrices $\boldsymbol{A}$, $\boldsymbol{C}$, $\boldsymbol{E}$ and the $N_{FE} \times K$ mode shape matrices $\boldsymbol{\Phi}_x$, $\boldsymbol{\Phi}_y$, $\boldsymbol{\Phi}_z$, $\boldsymbol{\Phi}_\phi$, $\boldsymbol{\Phi}_\theta$, $\boldsymbol{\Phi}_\psi$ calculate on a finite element grid defined by the $N_{FE} \times 1$ vectors $\boldsymbol{x}_{FE}$, $\boldsymbol{y}_{FE}$ and, optionally, $\boldsymbol{z}_{FE}$. Currently, two types of mode shapes are used in `SDPMflut`:

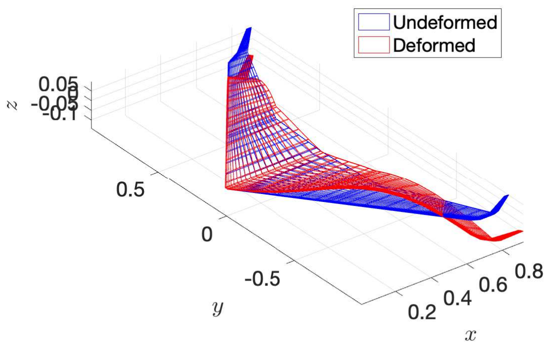
- Beam mode shapes: Translation and rotation mode shapes of beams with coordinates $\boldsymbol{x}_{FE}$, $\boldsymbol{y}_{FE}$, $\boldsymbol{z}_{FE}$. This type of mode shape is used in test case `T_tail`.
- Plate mode shapes: Translation and rotation mode shapes of flat plates with coordinates $\boldsymbol{x}_{FE}$, $\boldsymbol{y}_{FE}$, $\boldsymbol{z}_{FE}$, with $\boldsymbol{z}_{FE} = 0.0$. This type of mode shape is used in all other flexible flutter test cases.

The mode shapes in all the test cases are normalized such that the mass matrix $\boldsymbol{A}$ is the unit matrix. The damping and stiffness matrices are calculated by functions `FE_matrices` or `FE_matrices_beam` in package `FEmodes.py`, which takes the following inputs:

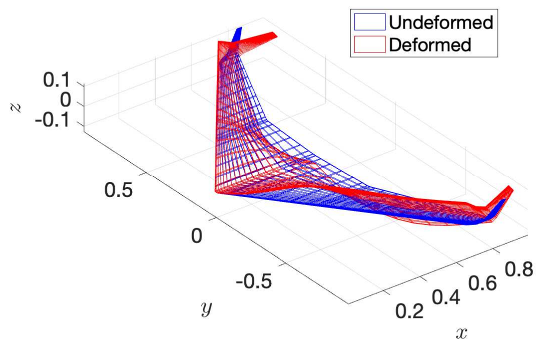
- `fname`: The name of a `.mat` file in the current working directory that contains the structural mass and stiffness matrices, $\boldsymbol{A}$ and $\boldsymbol{E}$, as well as $\boldsymbol{\Phi}_x$, $\boldsymbol{\Phi}_y$, $\boldsymbol{\Phi}_z$, $\boldsymbol{\Phi}_\phi$, $\boldsymbol{\Phi}_\theta$, $\boldsymbol{\Phi}_\psi$, $\boldsymbol{x}_{FE}$, $\boldsymbol{y}_{FE}$ and $\boldsymbol{z}_{FE}$. Currently, the only means for inputting structural information to `SDPMflut` is by means of Matlab `.mat` files. Future versions will introduce other input methods.
- `nmodes`: The number of modes, K, selected by the user for the flutter analysis. The maximum number of modes is the number of modes contained in `fname`. Fewer modes can be specified.



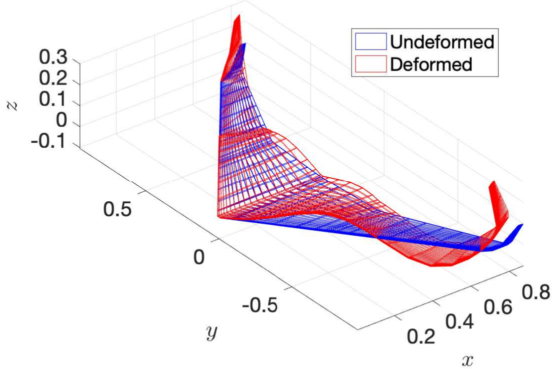
(a) Mode 1



(b) Mode 2



(c) Mode 3

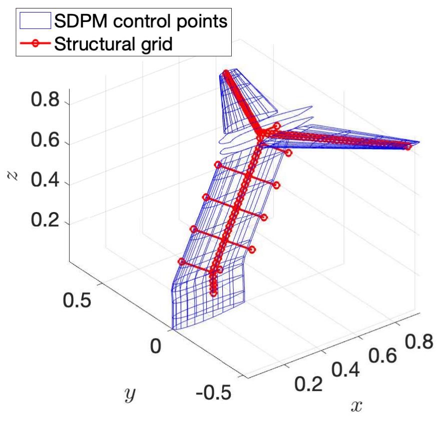


(d) Mode 3

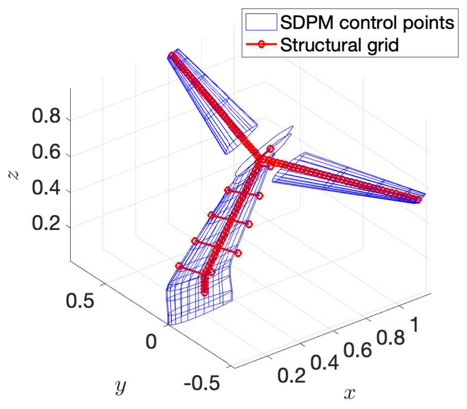
Figure 5: First four mode shapes of the NASA TMX 72799 wing with heavy winglet.

- `zeta0`: A 1D array with `nmodes` elements containing the values of the structural damping ratios corresponding to each mode.

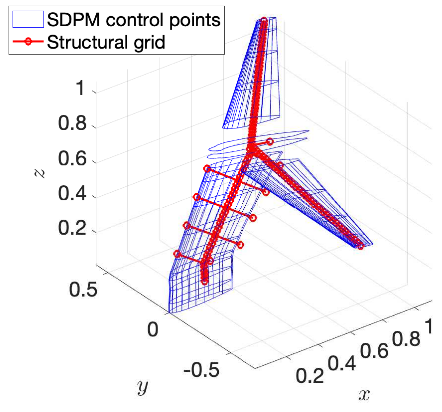
Once the modal matrices and mode shapes have been acquired, the latter are interpolated onto the SDPM grids of all the bodies in the flow using function `SDPMmodeinterp` in package `FEmodes.py`, except for test case `T_tail/flutter_SDPM_Ttail.py` for which function `ttailmodesinterp` in package `VanZylTtail.py` is used instead. Currently, function `SDPMmodeinterp` works only for single wings with flat plate mode shapes. The wings are cantilevered at the root so that the mode shapes are defined on the right half-wing, on a x-y grid with N_{FE} nodes. These mode shapes are interpolated onto the mean surface (camber surface) of the right-hand side of the SDPM grid using 2D scattered data cubic interpolation, with respect to the x and y coordinates of the mean surface. The interpolated mode shapes are mirrored to the left half-wing and applied to both the upper and lower surfaces. Figure 5 plots the first four mode shapes of the NASA TMX 72799 wing with heavy winglet, corresponding to test case `flutter_SDPM_NASATMX72799.py`



(a) Mode 1



(b) Mode 2



(c) Mode 3

Figure 6: T-tail deformed parallel to the first three mode shapes[3].

with option `winglet=3`. It can be seen that the interpolated mode shapes are symmetric across the $y = 0$ plane. Mode 1 is mostly first wing bending, mode 2 second wing bending with a bit of first wing torsion, mode 3 second wing torsion with a bit of winglet bending and mode four third wing bending with a bit of torsion.

Function `ttailmodesinterp` works for all the components of the Van Zyl T-tail (fin, left and right horizontal tailplanes and fin fairing) and beam mode shapes. The T-tail structural model is cantilevered at the root of the fin beam. The mode shapes are interpolated separately onto the fin, tailplanes and fin fairing using radial basis functions. The interpolation is carried out directly onto the two surfaces of each body, using x and z control point coordinates for the fin and fin fairing and x and y control point coordinates for the two tailplanes. Only the nodes of the structural model that are relevant to each body are used in the interpolations. Figure 6 plots the T-tail deformed parallel to the first three modes, comparing the deformed structural grid to the deformed SDPM grid.

7 Input file

Every `SDPMflut` input file starts with setting the value of `install_dir` to tell Python where to look for the files in ./Common. Next, the necessary libraries and data types are imported. For the SDPM solver:

```
# Input installation directory
install_dir=r"/Users/Username/Documents/Python/SDPMflut_v0.5/Common/"
# Import libraries and packages
import numpy as np
import matplotlib.pyplot as plt
import sys
sys.path.append(install_dir)
from SDPMgeometry import SDPMgeometry_trap_fun
import flutsol
import FEmodes
import SDPMcalcs
# Acquire SDPMflut trap and body data types
tp_trap, tp_body, _=SDPMcalcs.SDPMdtypes()
```

Additional libraries can be imported, depending on the needs of each project. For the DLM solver, the packages

```
from DLMgeometry import DLMgeometry_trap_fun
from SDPMcalcs import modeshape_assemble
import DLMcalcs
# Create DLM data types
tp_trap, tp_body, _=DLMcalcs.DLMdtypes()
```

are imported instead of the corresponding SDPM packages.

The next step is to define the flight conditions for the `SDPMflut` analysis. The minimum set of variables that need to be defined are:

- `Machdata`: The free stream Mach number.
- `rhodata`: The free stream density.
- `alpha0`: The steady (or mean) angle of attack
- `beta0`: The steady (or mean) angle of sideslip

Each of these variables can be a scalar or an array with `nruns` elements, where `nruns` is the desired number of runs of the `SDPMflut` analysis at different flight conditions. Note that different runs can be defined not only in terms of flight conditions, but also in terms of some geometric or structural parameter(s), such as the tailplane incidence in example `T_tail`. The units used in `SDPMflut` can be SI or imperial, as long as they are consistent. All angles must given in rad and all frequencies in rad/s.

Next, the geometries of all the bodies present in the flow are defined. The first step is to set the number of bodies and to initialize the `body` structured array:

```
# Set number of bodies
nbody=1
# Initialize body struct array
body=np.zeros(nbody,dtype=tp_body)
```

Then, the geometry of each body must be inputted. For the SDPM:

```
# Input first body
ibody=0         # Index of body
name='wing'     # Name of body
# Choose numbers of panels for this wing and its wake
nhalf=10        # Number of spanwise panels per half-wing.
m=20            # Number of chordwise panels
nchords=10      # Set length of wake in chord lengths
# Calculate number of chordwise wake rings
mw=m*nchords
# Set number of trapezoidal sections for this wing
ntrap=1
# Initialize trapezoidal section struct array
trap=np.zeros(ntrap,dtype=tp_trap)
```

These lines tell the software which body is being defined, what is its desired grid size and how many trapezoidal sections are necessary for its definition. The DLM does not require `nchords` and `mw`. The array `trap` is initialized and values for all of its fields must be given, as defined in section 5.1. Then, the geometric data of each trapezoidal section, are inputted into the `trap` array.

```
# Arrange all data into trapezoidal sections
trap[0]=np.array([(c0,xledist,bhalf,lamda,LamdaLE,roottwist,tiptwist,
     twistcent,dihedral,rootairfoil,rootairfoil_params,tipairfoil,
     tipairfoil_params)],dtype=tp_trap)
```

The panel aspect ratio, `panelAR`, can be defined for a complete body or for a trapezoidal section. For a complete wing, it is recommended that the panel aspect ratio should not take values less than 0.1 for the SDPM and 0.25 for the DLM. The input files in the test cases check for this using:

```
# Calculate panel aspect ratio
panelAR=(c0/m)/(bhalf/nhalf)
if panelAR < 0.1:
    sys.exit('Panel aspect ratio too low. Increase n or decrease m.')
```

The user can remove these lines, they only serve as a warning.

Finally, the SDPM grid for body `ibody` is generated:

```
# Calculate vertices of wing panels
body=SDPMgeometry_trap_fun(body,ibody,m,mw,nhalf,mirroredwing,linchord,
    linspan,trap,name,dir_tau,rollpitchyaw,rollpitchyaw_cent,lexyz,nmin)
```

where all of the inputs to function SDPMgeometry_trap_fun must be set beforehand, as detailed in section 5.1. Once the SDPM grids for all bodies have been generated, the software will calculate the numbers and indices of panels, spanwise panels and wake panels in all bodies stored in struct array body and create structured array allbodies:

```
# Assemble the indices of the body panels, spanwise body panels, wake
# panels etc. for all bodies.
allbodies=SDPMcalcs.allbodyindex(body)
```

Array allbodies is of type tp_allbodies and contains only outputs of the software. It is a placeholder for information taken from all the bodies present in the flow. Six of its fields contain flow information that is not available elsewhere:

- barphix0, barphiy0,barphiz0: Normalized steady perturbation velocities at the control points of all the panels of all the bodies.
- baruc0, barvc0,barwc0: Normalized steady total velocities (including the free stream) at the control points of all the panels of all the bodies.

The other fields in allbodies concerning flow quantities (e.g. cp0) can also be found in the elements of body.

The DLM grid is generated using:

```
body=DLMgeometry_trap_fun(body,ibody,m,nhalf,mirroredwing,linchord,
     linspan,trap,name,rollpitchyaw,rollpitchyaw_cent,lexyz,nmin,bchar)
```

Note that linchord and linspan must both be set to 1. The allbodies array is generated using

```
# Assemble the indices of the body panels, spanwise body panels, wake
# panels etc. for all bodies.
allbodies=DLMcalcs.allbodyindexDLM(body,bchar)
```

and contains slightly different information to the same array used by the SDPM.

The next step is to carry out the desired analysis: there are three analysis types:

1. Calculation of steady aerodynamic pressures (pressure jumps for the DLM), surface velocities (for the SDPM only) and loads on the control points of the panels of all the bodies.
2. Calculation of unsteady aerodynamic pressures (pressure jumps for the DLM), surface velocities (for the SDPM only) and loads on the control points of the panels of all the bodies. This analysis requires the steady aerodynamic calculation for the SDPM but not for the DLM.
3. Calculation of the flutter speed and frequency. This analysis requires the unsteady aerodynamic calculation.

Each of the analysis will be demonstrated separately by means of the relevant test cases that come with the software.

7.1 Steady aerodynamic analysis

Once the flight conditions and body geometries have been defined, steady aerodynamic analysis is carried out by typing:

```
# Calculate steady aerodynamic pressures and loads
body,allbodies,Aphi,Bphi,Cphi,barUinf,barVinf,barWinf=
         SDPMcalcs.steadysolve(body,allbodies,cp_order,Mach,beta,alpha0,
         beta0,xf0,yf0,zf0,install_dir)
```

where `xf0`, `yf0`, `zf0` are the Cartesian coordinates of a point around which rotations and/or moments are to be calculated. This point can be the centre of gravity or any rotation axis; `xf0`, `yf0`, `zf0` can all be set to 0.0 if no rotations or moments are to be calculated. Function `SDPMcalcs.steadysolve` calculates:

1. `barUinf,barVinf,barWinf`: the non-dimensional free stream components, $\bar{U}_\infty$, $\bar{V}_\infty$, $\bar{W}_\infty$, of equation 21
2. the Prandtl-Glauert transformation of equation 3 for all geometries
3. `Pe0,Pc`: the matrices $\boldsymbol{P}_e(0)$ in equation 15 and $\boldsymbol{P}_c$ in equation 6
4. `Aphi,Bphi,Cphi`: the steady aerodynamic influence coefficient matrices, $\boldsymbol{A}_\phi$, $\boldsymbol{B}_\phi$, $\boldsymbol{C}_\phi$, in equation 13
5. `mu0`: the steady doublet strength on the body panels, $\bar{\boldsymbol{\mu}}(0) = \boldsymbol{\mu}(0)/Q_\infty$, from equations 12, 13 and 14
6. `muw0`: the steady doublet strength on the wake panels, $\bar{\boldsymbol{\mu}}_w(0) = \boldsymbol{\mu}_w(0)/Q_\infty$, from equation 5 for $k = 0$
7. `barphix0,barphiy0,barphiz0`: the steady perturbation velocities on the body panels, $\bar{\boldsymbol{\phi}}_x(0) = \boldsymbol{\phi}_x(0)/Q_\infty$, $\bar{\boldsymbol{\phi}}_y(0) = \boldsymbol{\phi}_y(0)/Q_\infty$, $\bar{\boldsymbol{\phi}}_z(0) = \boldsymbol{\phi}_z(0)/Q_\infty$, from equations 16
8. `baruc0,barvc0,barwc0`: the steady total velocities on the body panels, $\bar{\boldsymbol{u}}(0) = \boldsymbol{u}(0)/Q_\infty$, $\bar{\boldsymbol{v}}(0) = \boldsymbol{v}(0)/Q_\infty$, $\bar{\boldsymbol{w}}(0) = \boldsymbol{w}(0)/Q_\infty$, from equation 17
9. `cp0`: the steady pressure coefficient on the body panels, $\boldsymbol{c}_{p_0}$, from equation 20
10. `Fx0,Fy0,Fz0`: the steady aerodynamic loads per dynamic pressure on the body panels, $\boldsymbol{F}_x(0)$, $\boldsymbol{F}_y(0)$, $\boldsymbol{F}_z(0)$, from equation 48

The function returns `barUinf,barVinf,barWinf` and `Aphi,Bphi,Cphi` directly to the input function's workspace so that they can be used in subsequent analyses. The variables `barphix0,barphiy0,barphiz0`, `baruc0,barvc0,barwc0`, `cp0`, `Fx0,Fy0,Fz0` and `Mx0,My0,Mz0` are stored in the respective fields of the `allbodies` structured array. For each of the bodies, the variables `mu0`, `muw0`, `cp0`, `Fx0,Fy0,Fz0` and `Mx0,My0,Mz0` are extracted, reshaped into $2m \times n$ matrices and stored in the respective fields of the respective element of the `body` structured array.

The `NASATM84367` test case is an example of a purely steady aerodynamic analysis of a wing at different Mach numbers and angles of attack. The input file is

```
steady_SDPM_NASATM84367.py
```

and the different flight conditions are entered as numpy arrays:

```
# Free stream Mach number. The last two test cases are highly transonic
Machdata=np.array([0.501, 0.499, 0.601, 0.601, 0.695, 0.695, 0.794,
      0.793])
# Mean angle of attack in degrees
alpha0data=np.array([0.0, 2.0, -2.0, 2.0, -2.0, 2.0, 2.0,
      -2.0])*np.pi/180.0
# Total number of runs
nruns=Machdata.size
```

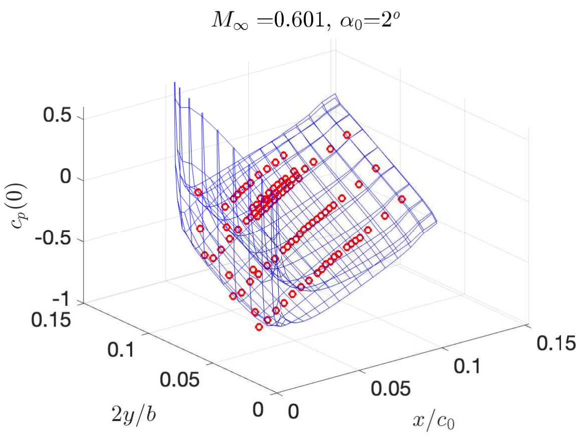
The experimental pressure measurements are loaded from the data file `dataNASATM84367`:

```
# Load experimental pressure data from .mat file
mat = scipy.io.loadmat("dataNASATM84367.mat")
```

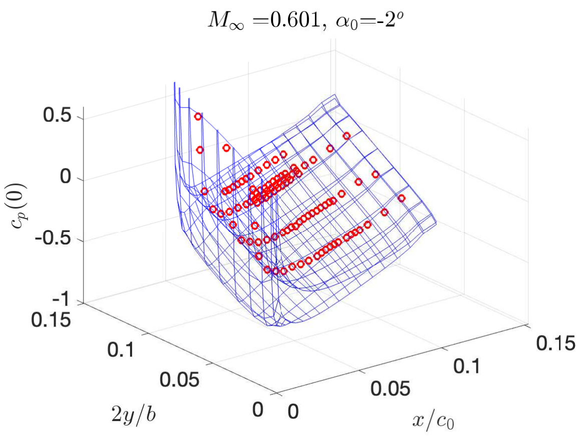
The geometry input is carried out as described in section 7. Then, `SDPMcalcs.steadysolve` is called for each of the `nruns` runs:

```
print('Calculating flutter solutions for all experimental test cases')
for irun in range (0,nruns):
    print('Simulating run '+str(irun+1))

    # Set Mach number of current run
    Mach=Machdata[irun]
    # Set mean angle of attack
    alpha0=alpha0data[irun]
    # Calculate subsonic compressibility factor
    beta=np.sqrt(1-Mach**2)

    # Calculate steady aerodynamic pressures and loads
    body,allbodies,Aphi,Bphi,Cphi,barUinf,barVinf,barWinf=
          SDPMcalcs.steadysolve(body,allbodies,cp_order,Mach,beta,alpha0,
          beta0,xf0,yf0,zf0,install_dir)

    fig, axx = plt.subplots(subplot_kw={"projection": "3d"})
    # Plot SDPM pressure predictions
    axx.plot_surface(body['Xc0'][0][:,body['n'][0]//2:body['n'][0]],
          body['Yc0'][0][:,body['n'][0]//2:body['n'][0]],
          body['cp0'][0][:,body['n'][0]//2:body['n'][0]],
          edgecolor='royalblue',alpha=0.1)
    # Plot experimental pressure measurements
    axx.scatter(mat['x0data'],mat['y0data'],mat['cp0data'][:,irun],
          marker='o',color='r')
    axx.set_proj_type('ortho')  # FOV = 0 deg
    axx.set_zlim(-1,0.6)
    axx.set_xlabel("$x/c_0$", labelpad=10)
    axx.set_ylabel("$2y/b$", labelpad=10)
    axx.set_zlabel("$c_p(0)$", labelpad=-1)
    axx.view_init(26, -120)
    plt.show()
# End for
```

All the lines after the call to `SDPMcalcs.steadysolve` plot the pressure distribution around the half-wing, stored in `body['cp0'][0]` against the x and y coordinates of the panel control points, `body['Xc0'][0]`, `body['Yc0'][0]`. Note that `body['Xc0'][0]`, `body['Yc0'][0]` describe a full-span wing geometry; the right half-wing is contained in the columns of these arrays that have indices from $n/2$ to $n-1$.



(a) $\alpha_0 = 2^\circ$



(b) $\alpha_0 = -2^\circ$

Figure 7: Steady pressure distributions for the NASATM8436 test case at $M_\infty = 0.6$

The pressure distributions calculated by the SDPM are compared to the experimental pressure measurements for each run. Pressure tappings were only installed on the upper surface of the wing, which is a suction surface for positive angles of attack and a pressure surface for negative angles of attack. Figure 7 plots the predicted and measured pressure distributions at $M_\infty = 0.6$ for $\alpha_0 = 2^\circ$ and $\alpha_0 = -2^\circ$[1]. The experimental pressures are denoted by the red circles, which lie on the suction side on figure 7(a) and on the pressure side on figure 7(b). The SDPM predictions are quite accurate at this Mach number; at $M_\infty = 0.8$ there is a strong shock wave on the suction side that cannot be modelled by the SDPM. For the DLM solver, the only significant difference is the function call:

```
    # Calculate steady aerodynamic pressures and loads
    body,allbodies,barUinf,barVinf,barWinf,Cx0,Cy0,Cz0,Cl0,Cm0,Cn0=
              DLMcalcs.steadysolveDLM(body,allbodies,Mach,alpha0,beta0,
              xf0,yf0,zf0,install_dir)
```

[1]The figures in this document were plotted in Matlab and not Matplotlib.

The aerodynamic loads per dynamic pressure (dimensions of length squared) are stored in `body['Fx0'][0]`, `body['Fy0'][0]`, `body['Fz0'][0]` as matrices with the same dimensions as `body['Xc0'][0]`, `body['Yc0'][0]`, `body['Zc0'][0]`. To obtain the aerodynamic loads in force units type:

```
Fx0_dimensional=body['Fx0'][0]*0.5*rhoinf*Qinf**2
Fy0_dimensional=body['Fy0'][0]*0.5*rhoinf*Qinf**2
Fz0_dimensional=body['Fz0'][0]*0.5*rhoinf*Qinf**2
```

where `rhoinf` is the free stream density, ρ_∞, and `Qinf` the free stream airspeed, Q_∞; both must be defined before the calculation. To obtain aerodynamic load coefficients in the x, y and z directions type:

```
CX0=body['Fx0'][0]/body['S'][0]
CY0=body['Fy0'][0]/body['S'][0]
CZ0=body['Fz0'][0]/body['S'][0]
```

The total lift and drag coefficients acting on the wing are obtained from:

```
CL=np.sum(CZ0)*np.cos(alpha0)-np.sum(CX0)*np.sin(alpha0)
CD=np.sum(CZ0)*np.sin(alpha0)+np.sum(CX0)*np.cos(alpha0)
```

7.2 Unsteady aerodynamic analysis

Unsteady aerodynamic analyses are carried out after the steady analysis. The only requirements are the definition of the motion kinematics of all the bodies and the input of a value for the reduced frequency, k, of the motion. The motion can be rigid translation and rotation around a given rotation centre or flexible and parallel to give mode shapes. Test case `NASATND344` is an example of the latter; a rectangular wing was placed in the wind tunnel and forced to oscillate in bending around its first bending mode shape, with imposed amplitude and frequency. The test case input file, `unsteady_SDPM_NASATND344.py`, includes

```
# Bending tip amplitude (m)
bendtip_amp=0.2*0.0254
# Bending phase (rad)
Phi_bend=0.0
```

where `bendtip_amp` is the bending oscillation amplitude at the wingtip and `Phi_bend` is the phase angle of the oscillation, such that `Phi_bend=0.0` means that the motion is a pure cosine function of time. In the frequency domain, the displacement of the wingtip in the z direction at frequency k is given by

```
bendtip_amp/2.0*np.exp(1j*Phi_bend)
```

The bending mode shape, $\mathbf{\Phi}_z$, is created on a structural grid with `mFE` chordwise and `nFE` spanwise points,

```
# Choose number of modes to include in the flutter calculation
nmodes=1    # The wing was forced to oscillate in the first bending mode
# Set up structural modal grid
mFE=30  # Set desired number of chordwise points
nFE=30  # Set desired number of spanwise points
# Acquire mode shapes (only modeshapesz and its derivative modeshapesRx
# are non-zero)
xxplot,yyplot,modeshapesx,modeshapesy,modeshapesz,modeshapesRx,
          modeshapesRy,modeshapesRz=modes_NASATND344(mFE,nFE)
```

such that $\mathbf{\Phi}_z$ is 0 at the root and 1 at the tip. It is assumed that there is no torsion, so that $\mathbf{\Phi}_z$ is constant in the chordwise direction. As only bending was imposed, $\mathbf{\Phi}_x = \mathbf{\Phi}_y = \mathbf{\Phi}_\psi = 0$. Consequently,

$$\mathbf{\Phi}_\phi = -\frac{\partial \mathbf{\Phi}_z}{\partial y},\ \mathbf{\Phi}_\theta = -\frac{\partial \mathbf{\Phi}_z}{\partial x}$$

Since $\mathbf{\Phi}_z$ is constant in the chordwise direction, $\mathbf{\Phi}_\theta = 0$, so that only $\mathbf{\Phi}_\phi$ and $\mathbf{\Phi}_z$ are non-zero. Function `modes_NASATND344` returns in $m_{FE}n_{FE} \times 1$ arrays:

- `xxplot,yyplot`: The x and y coordinates of the structural grid.
- `modeshapesx,modeshapesy,modeshapesz`: The translation components of the mode shape, $\mathbf{\Phi}_x$, $\mathbf{\Phi}_y$, $\mathbf{\Phi}_z$.
- `modeshapesRx,modeshapesRy,modeshapesRz`: The rotation components of the mode shape, $\mathbf{\Phi}_\phi$, $\mathbf{\Phi}_\theta = 0$, $\mathbf{\Phi}_\psi$.

Function `unsteady_SDPM_NASATND344.py` interpolates the components of the mode shape onto the SDPM grid, once have been acquired, using

```
# Interpolate mode shapes onto panel control points
body=FEmodes.SDPMmodeinterp(xxplot,yyplot,modeshapesx,modeshapesy,
      modeshapesz,modeshapesRx,modeshapesRy,modeshapesRz,body)
```

Function `FEmodes.SDPMmodeinterp` takes the x and y cartesian coordinates of the panel control points from `body['Xc0'][0]`, `body['Yc0'][0]`. The interpolation is carried out onto the mean surface (camber surface) of the right half-wing; it is then assigned to both the upper and lower surfaces and mirrored across the centreline of the wing. The interpolated mode shape components are $2mn \times 1$ arrays stored in struct array `body`:

- `body['Phi_xall'][0],body['Phi_yall'][0],body['Phi_zall'][0]`: The interpolated translation components of the mode shape, $\tilde{\mathbf{\Phi}}_x$, $\tilde{\mathbf{\Phi}}_y$, $\tilde{\mathbf{\Phi}}_z$.
- `body['Phi_phiall'][0],body['Phi_thetaall'][0],body['Phi_psiall'][0]`: The interpolated rotation components of the mode shape, $\tilde{\mathbf{\Phi}}_\phi$, $\tilde{\mathbf{\Phi}}_\theta$, $\tilde{\mathbf{\Phi}}_\psi$.

The next step in `unsteady_SDPM_NASATND344.py` is to concatenate the mode shape components for all the bodies:

```
# Assemble mode shapes for all bodies into global matrices
allbodies=SDPMcalcs.modeshape_assemble(body,allbodies,nmodes)
```

In the present case there is only one body so the only result of this operation is the copying of `body['Phi_xall'][0]`, `body['Phi_yall'][0]`, etc, into `allbodies['Phi_x'][0]`, `allbodies['Phi_y'][0]` etc.

For each of the experimental runs, the Mach number, reduced frequency, angle of attack, subsonic compressibility factor and effective angle of attack at the wingtip are set:

```
    # Set Mach number of current run
    Mach=Machdata[irun]
    # Set reduced frequency
    k=kdata[irun]
    # Set mean angle of attack
    alpha0=alpha0data[irun]
    # Calculate subsonic compressibility factor
    beta=np.sqrt(1-Mach**2);
    # Calculate effective angle of attack at the wingtip
    alpha_h=2.0/c0*k*bendtip_amp
```

where `irun` is the index of the current run. Next, the steady aerodynamic calculation is carried out

```
    # Calculate steady aerodynamic pressures and loads
    body,allbodies,Aphi,Bphi,Cphi,barUinf,barVinf,barWinf=
         SDPMcalcs.steadysolve(body,allbodies,cp_order,Mach,beta,alpha0,
         beta0,0.0,0.0,0.0,install_dir)
```

and the steady pressure distribution is plotted and compared to experimental data. Then, the unsteady aerodynamic analysis is carried out:

```
    # Calculate the unsteady pressure coefficients
    cp1,cp_0,cp_1,cp_2=SDPMcalcs.unsteadysolve_flex(body,allbodies,Aphi,
          Bphi,Cphi,barUinf,barVinf,barWinf,k,c0,Mach,beta,cp_order,
          install_dir)
```

Function `SDPMcalcs.unsteadysolve_flex` calculates the oscillatory pressure distribution for flexible motion using the mode shapes stored in array `body`. The function first calculates the non-dimensional frequency $\Omega = 2kM_\infty/c_0\beta$ and then the wake double strength decay matrix $\boldsymbol{P}_e(k)$ of equation 7, before computing the unsteady aerodynamic influence coefficient matrices using:

```
    # Calculate unsteady influence coefficient matrices
    Abarphi,Bbarphi,Cbarphi=unsteady_infcoef(body,allbodies,install_dir,
         Omega,Mach,Aphi,Bphi,Cphi)
```

where `Abarphi`, `Bbarphi`, `Cbarphi` are the matrices $\hat{\boldsymbol{A}}_\phi(k)$, $\hat{\boldsymbol{B}}_\phi(k)$, $\hat{\boldsymbol{C}}_\phi(k)$ appearing in equations 9. Then, it evaluates the matrices $\boldsymbol{K}(k)$, $\boldsymbol{K}_x(k)$, $\boldsymbol{K}_y(k)$, $\boldsymbol{K}_z(k)$ in equations 9 to 10, followed by the matrices $\bar{\boldsymbol{C}}_0(k)$ and $\bar{\boldsymbol{C}}_1(k)$ in equation 19. Finally, the function outputs the pressure derivative arrays `cp1,cp_0,cp_1,cp_2`, which have size `allbodies['allpanels']`$\times$`nmodes` and are given by

$$\begin{aligned}
\boldsymbol{c}_{p_0}(k) &= \boldsymbol{c}_{p_\phi}(k)+\boldsymbol{c}_{p_\theta}(k)+\boldsymbol{c}_{p_\psi}(k)\\
\boldsymbol{c}_{p_1}(k) &= \boldsymbol{c}_{p_{\dot{x}}}(k)+\boldsymbol{c}_{p_{\dot{y}}}(k)+\boldsymbol{c}_{p_{\dot{z}}}(k)+\boldsymbol{c}_{p_{\dot{\phi}}}(k)+\boldsymbol{c}_{p_{\dot{\theta}}}(k)+\boldsymbol{c}_{p_{\dot{\psi}}}(k)\\
\boldsymbol{c}_{p_2}(k) &= \boldsymbol{c}_{p_{\ddot{x}}}(k)+\boldsymbol{c}_{p_{\ddot{y}}}(k)+\boldsymbol{c}_{p_{\ddot{z}}}(k)
\end{aligned}$$

where $\boldsymbol{c}_{p_\phi}(k)$, $\boldsymbol{c}_{p_\theta}(k)$ etc. appear in equation 25. The function also outputs the total pressure derivative array `cp1`, which has the same dimensions and is $\boldsymbol{c}_p(k)$ and is given by

$$\boldsymbol{c}_{p_0}(k)+\mathrm{i}k\boldsymbol{c}_{p_1}(k)+(\mathrm{i}k)^2\,\boldsymbol{c}_{p_2}(k) \tag{50}$$

Finally, `unsteady_SDPM_NASATND344.py` calculates $\boldsymbol{c}_p(k)$ in equation 25 by multiplying `cp1` by the tip bending amplitude:

```
# Mulitply cp1 by tip bending amplitude and reshape to a matrix
cp1mat=bendtip_amp/2.0*np.exp(1j*Phi_bend)*np.reshape(cp1,
      (2*body['m'][0],body['n'][0]),order='C')
```

The array `cp1mat` has dimensions $2m\times n$ and is complex. It is the required oscillatory pressure distribution acting on the panel control points due to the imposed bending motion. The test case input file also calculates the pressure jump across the surface, $\Delta\boldsymbol{c}_p(k)$.

```
# Calculate pressure jump across the surface
 Dcp1=np.flipud(cp1mat[0:m,:])-cp1mat[m:2*m,:]
```

The final step is to plot the real and imaginary parts of the unsteady pressure distribution and to compare them to experimental data. The original reference gives this data only for the $M_\infty=0.24$ cases. Figure 8 plots the real and imaginary parts of the pressure distribution for $M_\infty=24$, $\alpha_0=5°$ and compares them to the experimental measurements. For all the Mach numbers, only the pressure jump distribution is given so this is what is plotted by `unsteady_SDPM_NASATND344.py`. Note that there are strong shock waves in the steady pressure distributions at $M_\infty=0.9$ and $M_\infty=0.7$ ($\alpha_0=5°$ case), which also have an effect on the unsteady pressure jump distributions.

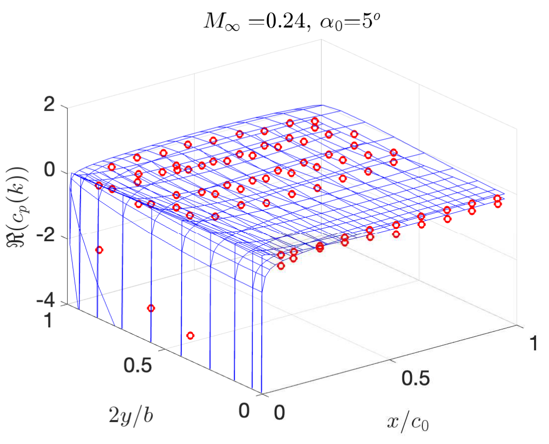
For the DLM solver, the steady calculation is carried out using

```
# Calculate steady aerodynamic pressures and loads
body,allbodies,barUinf,barVinf,barWinf,Cx0,Cy0,Cz0,Cl0,Cm0,Cn0=
                 DLMcalcs.steadysolveDLM(body,allbodies,Mach,alpha0,
                 beta0,0.0,0.0,0.0,install_dir)
```

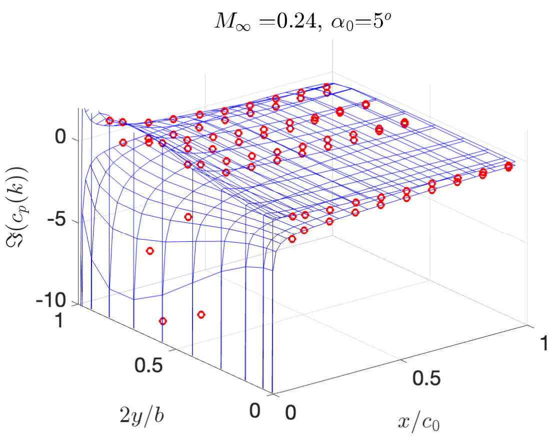
while the unsteady calculation is called by

```
# Calculate the unsteady pressure coefficients
Deltacp1,Deltacp_0,Deltacp_1=DLMcalcs.unsteadysolve_flexDLM(body,
               allbodies,k,Mach,install_dir)
```

It should be recalled that the DLM only calculates pressure jumps.



(a) $\Re(c_p(k))$



(b) $\Im(c_p(k))$

Figure 8: Real and imaginary parts of the pressure distribution for the NASATND344 test case at $M_\infty = 24$, $\alpha_0 = 5^\circ$

7.3 Calculation of aerodynamic stability derivatives

Test case `NACARML5` calculates the lateral aerodynamic stability derivatives of a Delta, a straight, and a swept wing and compares them to experimentally determined values given in [18, 19, 20, 21]. Each of the wings has its own input file, `unsteady_SDPM_delta.py`, `unsteady_SDPM_straight.py` and `unsteady_SDPM_swept.py`. The analysis starts as usual, with the input of the flight conditions, the description of the wing geometry and the generation of the SDPM grid. Since the lateral stability derivatives are of interest, the straight and swept wings feature wingtips that block the lateral flow through the wing. The SDPM grid is created as follows:

```
# Calculate vertices of wing panels
body=SDPMgeometry_trap_fun(body,ibody,m,mw,nhalf,mirroredwing,linchord,
    linspan,trap,name,dir_tau,rollpitchyaw,rollpitchyaw_cent,lexyz,nmin)
# Calculate vertices of wingtip panels
body=makewingtips(body,ibody,mirroredwing)
```

Function `makewingtips` adds one or two new elements in structured array body, containing the SDPM grid coordinates of left and right wingtips attached to `body[ibody]`. The number of wingtips depends on the value of `mirroredwing`; if `mirroredwing=-1` a left wingtip is added, if `mirroredwing=1` a right wingtip is added and if if `mirroredwing=2` both left and right wingtips are added. Figure 9 plots an example of a left wingtip added to the straight wing. Wingtip objects are identical in structure to all other `body` objects but they do not shed a wake, such that `mw=0`. The wingtip vertex matrices `Xp0`, `Yp0`, `Zp0` have $2m + 1$ lines, where m is the number of chordwise panels in the parent wing, and 3 columns:

- : Left wingtip: lower surface, mid surface and upper surface
- Right wingtip: upper surface, mid surface and lower surface

Consequently, wingtips have $2m$ chordwise and 2 heightwise panels. Aside from not shedding wakes, they are treated like all other elements of the `body` struct array.

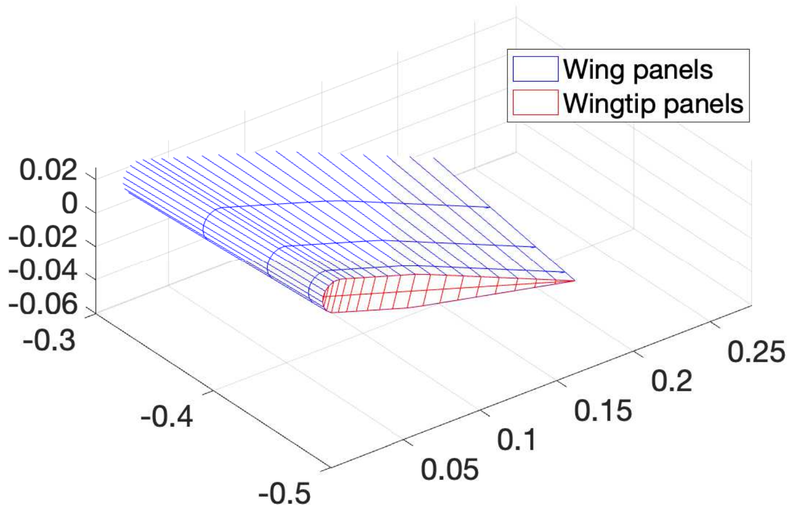


Figure 9: Wing and wingtip panels

The `NACARML5` test cases calculate all the aerodynamic stability derivatives at a single reduced frequency, $k_b = \omega b/2Q_\infty = 0.23$, where $b/2$ is the half-span, and a range of steady pitch angles. In order to simulate as closely as possible the experiments, the free stream angles of attack and sideslip are set to zero but the wing itself is rotated to the desired pitch angle, around the quarter of the mean aerodynamic chord. The rotation is carried by calling `SDPMgeometry_trap_fun` after setting

```
# Recreate the body at the current pitch angle value
# Define roll, pitch and yaw angles
rollpitchyaw=np.array([0, pitchdata[irun], 0])
# Define roll, pitch and yaw centre (x,y,z position of rotation centre)
rollpitchyaw_cent=np.array([xf0, yf0, zf0])
```

thus regenerating the SDPM grid at the current pitch angle, `pitchdata[irun]`. Then, the wingtips are recalculated to fit the new wing grid and $SDPMcalcs.steadysolve$ is called to solve the steady flow at the current pitch angle. Recall that `SDPMflut` calculates forces and moments in Newtons per Pascal and Newton meters per Pascal respectively. These are converted to steady load coefficients by

```
# Calculate steady aerodynamic load coefficients on the panels
CD[irun]=np.sum(body['Fx0'][0])/Sref
CY[irun]=np.sum(body['Fy0'][0])/Sref
CL[irun]=np.sum(body['Fz0'][0])/Sref
```

```
    Cl[irun]=np.sum(body['Mx0'][0])/Sref/bref
    Cm[irun]=np.sum(body['My0'][0])/Sref/cref
    Cn[irun]=np.sum(body['Mz0'][0])/Sref/bref
```

where `Sref` is the reference area, `bref` the reference span, `cref` the reference chord, `CD` is the drag coefficient, `CY` the sideforce coefficient, `CL` the lift coefficient, `Cl` the rolling moment coefficient, `Cm` the pitching moment coefficient and `Cn` the yawing moment coefficient.

For compatibility with the rest of the code, k_b is converted to the reduced frequency based on the half-chord

```
    # Convert reduced frequency based on half-span
    # to reduced frequency based on half-chord
    k=kvec[ik]/bhalf*c0/2.0
```

The derivatives of the pressure with respect to the degrees of freedom are evaluated using equations 81 to 95. Both the pressure derivatives and the aerodynamic stability derivatives are calculated by function `SDPMcalcs.aerostabderiv`:

```
    # Calculate aerodynamic stability derivatives
    stabder=SDPMcalcs.aerostabderiv(body,allbodies,Aphi,Bphi,Cphi,barUinf,
       barVinf,barWinf,k,c0,Mach,beta,cp_order,xf0,yf0,zf0,Sref,bref,
       cref,install_dir)
```

The aerodynamic stability derivatives are stored in structure array `stabder`, of type `tp_stabder`. This array contains 21 longitudinal derivatives, from `CXu` to `Cmqdot`, and 24 lateral derivatives, from `CYv` to `Cnrdot`. Given the definitions of $\bar{v}(k)$ and $\bar{\dot{v}}(k)$ in appendix A.2, it can be shown that

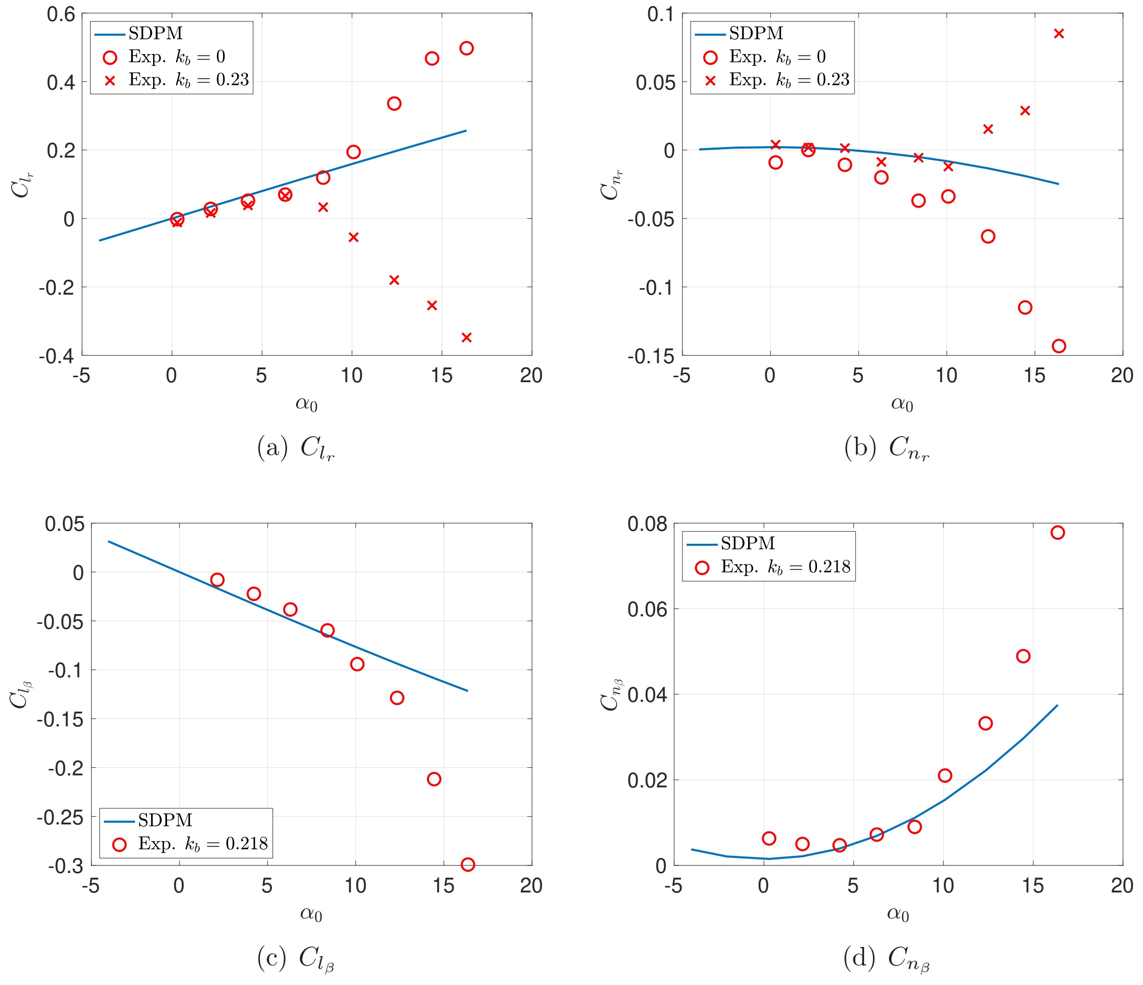
$$C_{Y_\beta}=-C_{Y_v},\ C_{Y_{\dot\beta}}=-C_{Y_{\dot v}},\ C_{l_\beta}=-C_{l_v},\ C_{l_{\dot\beta}}=-C_{l_{\dot v}},\ C_{n_\beta}=-C_{n_v},\ C_{n_{\dot\beta}}=-C_{n_{\dot v}}$$

As an example, figure 10 plots the variation of the roll and yaw aerodynamic stability derivatives with respect to $\bar{r}$ and β with steady pitch angle for the straight wing. The agreement between the SDPM predictions and experimental data is very good up to pitch angle values of around 8°. At higher angles, viscous phenomena become very important and the inviscid SDPM cannot predict them. For compatibility with the experimental data, only the real parts of the aerodynamic derivatives are plotted; the imaginary parts are very small compared to the real parts at this value of the reduced frequency.

The measured aerodynamic stability derivatives with respect to accelerations $\dot{r}$ and $\dot{\beta}$ are quite flat at low pitch angles; it is only at the higher pitch angles that significant values are observed. The SDPM predicts very small values for these derivatives at all pitch angles. Finally, it should be noted that the sideslip derivatives predicted by the SDPM are less accurate than the roll and yaw derivatives.

7.4 Flutter analysis for flexible wings

The contents of an input file for flutter analysis are fairly similar to those of an input file for unsteady analysis. There are three flutter cases of flexible structures in the current `SDPM` distribution:



(a) C_{l_r} (b) C_{n_r}

(c) C_{l_β} (d) C_{n_β}

Figure 10: Variation of aerodynamic stability derivatives with steady pitch angle for the straight wing

- `flutter_SDPM_AGARD.py`
- `flutter_SDPM_NASATMX72799.py`
- `flutter_SDPM_Ttail.py`

The input files starts as usual but the `flutsol` package must also be imported. The definition of flight conditions and the input of the geometry are carried out as usual. The structural model is loaded from a `.mat` file, whose name is stored in `fname`, e.g.

```
# File name of Matlab mat file that contains the structural model
fname='modes_AGARD_Q4EPM.mat'
# Choose number of modes to include in the flutter calculation
nmodes=5    # Cannot exceed number of modes in FE model
zeta0=0.02*np.ones(nmodes) # Structural damping ratios
```

```
# Parameter to determine if the structural model concerns a half wing or a
# full wing.
halfwing=1 # halfwing=1: half-wing. halfwing=0: full wing
```

File `modes_AGARD_Q4EPM.mat` contains the `nmodes`×`nmodes` arrays `Mmodal` and `Kmodal`, the `mFE*nFE`× coordinate arrays `xxplot`, `xxplot`, and the `mFE*nFE`×`nmodes` mode shape arrays `modeshapesx`, `modeshapesy`, `modeshapesz`, `modeshapesRx`, `modeshapesRy`, `modeshapesRz`. The file is read using:

```
# Acquire structural matrices and mode shapes
A, C, E, wn, xxplot, yyplot, zzplot, modeshapesx, modeshapesy, modeshapesz,
    modeshapesRx, modeshapesRy, modeshapesRz=
                    FEmodes.FE_matrices(fname,zeta0,nmodes)
```

Function `FEmodes.FE_matrices` also truncates the modal matrices and mode shapes to the first `nmodes` modes, sets the off-diagonal terms of the modal matrices to exactly zero[2], calculates the damping matrix given the wind-off structural damping ratios `zeta0`, and calculates the first `nmodes` natural frequencies, outputting them to array `wn`.

The user must also define the reduced frequencies and airspeeds at which to carry out the flutter analysis, e.g.:

```
# Select reduced frequency values
kvec=np.array([0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 1.0, 2.0])

# Select airspeed range in m/s
Uv=np.linspace(100,400,num=101)
```

The values in `kvec` and `Uv` must be informed by the physics of the problem. The minimum value in $k = 0.001$ is probably adequate for most applications but the maximum value depends on the highest wind-off natural frequency, lowest airspeed and characteristic chord length. In other words:

```
kmax=np.max(wn)*c0/2.0/np.min(Uv)
```

This calculation is not carried out by `SDPMflut` because the user may wish to select an even higher maximum value for k. The reduced frequency values must be denser towards the lowest range; flutter in aircraft occurs generally at $k < 1$ and often at $k < 0.4$. The airspeed range in `Uv` must be selected by trial and error, unless the flutter airspeed is known or suspected. For a complete aircraft, the entire speed range in the flight envelope can be assigned to `Uv`.

The only other difference between a flutter analysis input file and an unsteady analysis input file is the calling of the flutter solution function, e.g.:

```
    Uflut,freqflut,kflut,dynpressflut,omega,zeta=
            flutsol.flutsolve_flex(body,allbodies,kvec,Uv,
            nmodes,Aphi,Bphi,Cphi,barUinf,barVinf,barWinf,c0,
            Mach,beta,cp_order,A,C,E,rho,wn,halfwing,install_dir)
```

[2]The modal matrices obtained from finite element packages are not always exactly diagonal, the off-diagonal terms may be non-zero but very small.

Function `flutsol.flutsolve_flex` starts with calculating the steady generalized aerodynamic load vector, $\boldsymbol{Q}_0(0)$, given by

$$\boldsymbol{Q}_0(0) = -\left(\tilde{\boldsymbol{\Phi}}_x^T \boldsymbol{F}_x(0) + \tilde{\boldsymbol{\Phi}}_y^T \boldsymbol{F}_y(0) + \tilde{\boldsymbol{\Phi}}_z^T \boldsymbol{F}_z(0)\right)$$

where $\boldsymbol{F}_x(0)$, $\boldsymbol{F}_y(0)$, $\boldsymbol{F}_z(0)$ are defined in equation 48. If the structural model is a half-wing, it will divide $\boldsymbol{Q}_0(0)$ by two so that only the aerodynamic loads from one half-wing will be included in the aeroelastic equation. Currently, $\boldsymbol{Q}_0(0)$ is not used in any of the test cases included with `SDPMflut` but it could be used to carry out steady aeroelastic analysis by solving the steady version of equation 28

$$\left(\boldsymbol{E} - \frac{1}{2}\rho_\infty Q_F^2 \boldsymbol{Q}_0(0)\right)\boldsymbol{q}(0) = \frac{1}{2}\rho_\infty Q_F^2 \boldsymbol{Q}_s(0)$$

for the modal displacements $\boldsymbol{q}(0)$ induced by the steady aerodynamic loads. The procedure can also be iterative, deforming the SDPM geometry by $\tilde{\boldsymbol{\Phi}}_x q(0)$, $\tilde{\boldsymbol{\Phi}}_y q(0)$, $\tilde{\boldsymbol{\Phi}}_z q(0)$ and then re-calculating all the steady and unsteady aerodynamic influence coefficients and the generalized aerodynamic load vector and matrices.

The next step in `flutsol.flutsolve_flex` is to call

```
        # Calculate the unsteady pressure coefficients
        cp1,cp_0,cp_1,cp_2=SDPMcalcs.unsteadysolve_flex(body,allbodies,
                Aphi,Bphi,Cphi,barUinf,barVinf,barWinf,k,c0,Mach,beta,
                cp_order,install_dir)
```

to obtain the unsteady pressures and to calculate $\boldsymbol{Q}_0(k)$, $\boldsymbol{Q}_1(k)$, $\boldsymbol{Q}_2(k)$ using equations 29 and 30, for all specified values of the reduced frequency k. Once the loop for k has finished executing, if `halfwing=1`, all generalized aerodynamic load matrices are divided by 2. Then, the determinant iteration procedure of equation 31 is applied to evaluate the wind-on natural frequencies and damping ratios at all selected airspeeds:

```
    # Calculate eigenvalues using determinant iteration
    eigvals=detiterfun(A,C,E,Q_0,Q_1,Q_2,kvec,Uv,c0/2,rho,wn)
    # Calculate natural frequencies and damping ratios
    omega=np.absolute(eigvals)
    zeta=-eigvals.real/np.absolute(eigvals)
```

where `omega` is a `nmodes`×`len(Uv)` array storing the natural frequencies, $\omega_n = |\lambda|$, and `zeta` is an array of the same size storing the damping ratios,

$$\zeta_n = \frac{-\Re(\lambda)}{|\lambda|}$$

If any of the damping ratios become negative within the airspeed range, a function will be called to pinpoint exactly the flutter condition by solving the determinant problem of equation 32:

```
    # Calculate exact flutter velocity and frequency
    Uflut,freqflut=flutfind(A,C,E,Q_0,Q_1,Q_2,kvec,Uini,c0/2.0,rho,wini)
    # Calculate flutter reduced frequency
    kflut=freqflut*c0/2.0/Uflut
    # Calculate flutter dynamic pressure
    dynpressflut=1/2.0*rho*Uflut**2.0
```

where `Uini` and `wini` are initial guesses for the flutter velocity and frequency obtained from the crossing of `zeta` to negative values. The converged values for the flutter speed and frequency are stored in `Uflut,freqflut` respectively. The function also calculates the reduced flutter frequency, `kflut`, and the flutter dynamic pressure, `dynpressflut` before returning.

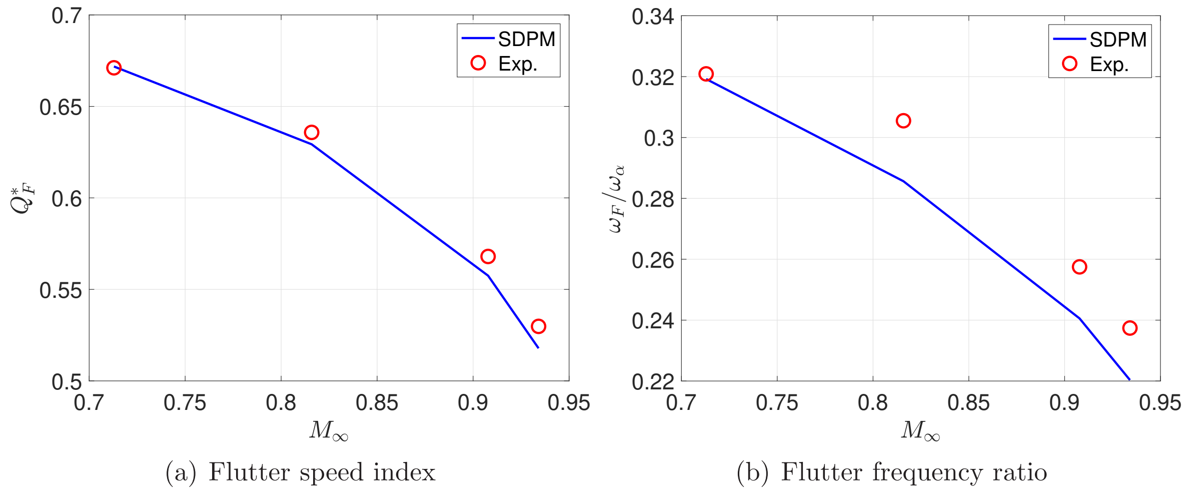


Figure 11: Variation of flutter speed index and flutter frequency ratio with Mach number for the NASATMX72799 test case with heavy winglets

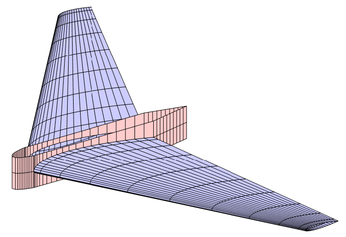
Finally, the input file calculates the flutter speed index,

$$Q_F^* = \frac{Q_F}{(c_0/2)\omega_\alpha\sqrt{\mu}}$$

where ω_α is usually the first wind-off torsional natural frequency and μ is the mass ratio, i.e. the ratio of a mass of air enclosing a body to the mass of the body. Then, it plots the flutter results for all the runs. For example, figure 11 plots the variation of flutter speed index and flutter frequency ratio with Mach number for the NASATMX72799 test case with heavy winglets.

The flutter solution function for the DLM is

```
# Calculate flutter solution for flexible motion
Uflut,freqflut,kflut,dynpressflut,omega,zeta=flutsol.flutsolve_flexDLM(body,
    allbodies,kvec,Uv,nmodes,Mach,A,C,E,rho,wn,halfwing,install_dir)
```


(a) T-tail fin fairing and horizontal tailplanes

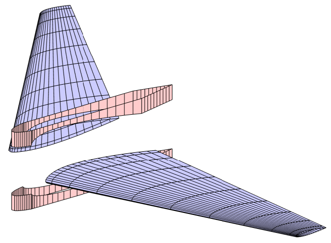

(b) Exploded view

Figure 12: SDPM grids for the fin fairing and horizontal taiplanes

The flutter speed can be calculated using either function `pkmethod` or `detiterfun`. At the moment, `pkmethod` has a bug so that `detiterfun` is preferred for most cases.

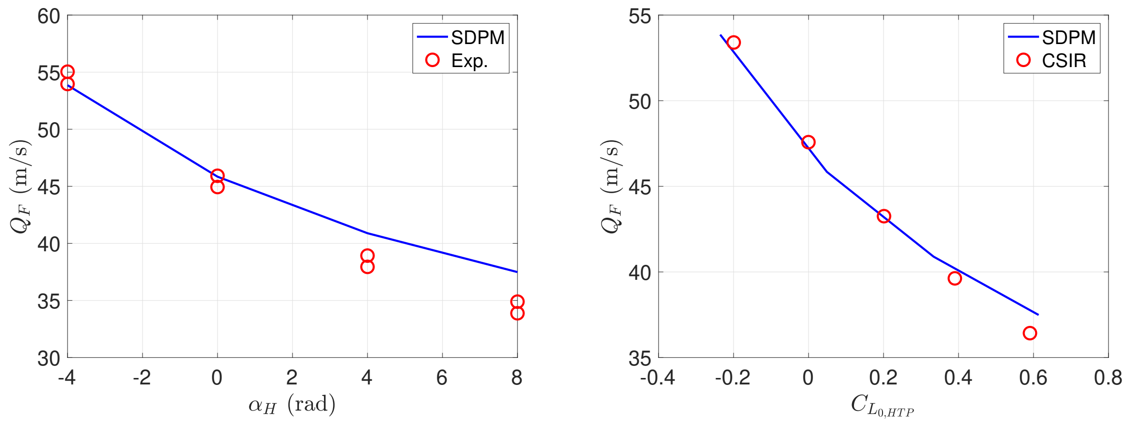
The `T_tail` case is particular in that there are many bodies in the flow and, more importantly, the horizontal tailplane is in contact with the fin fairing. When using panel methods, bodies that are in contact but do not share exactly the same contact vertices can cause significant numerical instabilities. The solution adopted in this test case is to create the fin fairing SDPM grid after creating that of the two tailplanes, such that the former shares the root vertices of the latter. As the angle of incidence of the horizontal tailplane varies between runs, the geometry is finalized inside the loop for `irun`; only the SDPM grid of the vertical fin is created before this loop. Figure12 plots the SDPM grids for the horizontal tailplanes and fin fairing only, as well as an exploded view of these elements. It can be seen that the fin fairing is split into two bodies, the upper and lower fairings, so that its grid can be accommodated into 2D arrays. Each half of the fairing has $2m$ chordwise panels on the the right side and another $2m$ on the left, while the horizontal tailplanes have m chordwise panels on the upper surface and m on the lower. The right half of the upper fairing shares m vertices with the upper surface of the right tailplane and the left half of the upper fairing shares m vertices with the upper surface of the left tailplane. Similarly, lower fairing shares vertices with the lower halves of the two tailplanes.

The other specificity of the `T_tail` test case is that it uses a dedicated mode interpolation function, instead of the one found in `Common`, that is:

```
# Interpolate mode shapes
body=VanZylTtail.ttailmodesinterp(body,z_root_tip,nmodes,xxplot,yyplot,
          zzplot,modeshapesx,modeshapesy,modeshapesz,modeshapesRx,
          modeshapesRy,modeshapesRz)
```

There are two reasons for this specificity:

- The mode shapes are beam modes; when interpolating them onto surfaces there is quite a lot of extrapolation. This means that radial basis function interpolation is necessary to avoid NaNs when using SciPy.
- Different parts of the mode shapes are used when interpolating on different bodies; the nodes of the mode shape lying on $y = 0$ are used for the fin and fin fairing while the nodes lying on $y \neq 0$ are used for the tailplanes.



(a) Flutter speed (b) Steady tailplane lift at flutter

Figure 13: Variation of flutter speed with tailplane incidence and steady tailplane lift at flutter for the Van Zyl T-tail test case

The rest of the `flutter_SDPM_Ttail.py` input file is similar to the AGARD and NASA TMX72799 input files, except that the experimental flutter frequencies are not given in the original references. Figure 13 plots the variation of the dimensional flutter speed with tailplane incidence, as well as its variation with steady tailplane lift coefficient. Note that the CSIR data plotted in figure 13(b) are not experimental; they were obtained using a modelling method by [28].

The DLM solution for the `T_tail` test case— is independent of the incidence angle of the horizontal tailplane. Consequently, only one geometric model is generated for all pitch angles. The inability of the DLM to model the effect of tailplane incidence on flutter speed for T-tails is a well-known limitation of the method [28].

7.5 Flutter analysis for pitching and plunging wings

The three test cases in the `PAPA` folder are rigid rectangular wings of exactly the same dimensions but with different airfoil sections, suspended from the same Pitching and Plunging Apparatus. The input files are identical except for the wind tunnel test conditions, airfoil sections and mass/stiffness characteristics. The difference with the flexible flutter test cases discussed in section 7.4 is that dedicated functions are used for calculating the unsteady aerodynamic pressures and flutter solutions. The normalized upwash due to the motion is given by a simplified version of equations 22,

$$
\begin{aligned}
\bar{\boldsymbol{u}}_m(k) &= -\frac{q(k)}{Q_\infty}(\boldsymbol{z}_c - z_f) \\
\bar{\boldsymbol{v}}_m(k) &= 0 \\
\bar{\boldsymbol{w}}_m(k) &= \theta(k) + \frac{q(k)}{Q_\infty}(\boldsymbol{x}_c - x_f) - \frac{w(k)}{Q_\infty}
\end{aligned}
\tag{51}
$$

where all the terms due to longitudinal, lateral, roll and yaw motion have been removed and x_f, y_f, z_f is the position of the pitch axis. Consequently, the unsteady pressure is given by a shorter version of equations 24, namely

$$
\begin{aligned}
\boldsymbol{c}_p(k) &= \boldsymbol{c}_{p_w}(k)w(k) + \boldsymbol{c}_{p_\theta}(k)\theta(k) + \boldsymbol{c}_{p_q}(k)q(k) + \mathrm{i}k\boldsymbol{c}_{p_{\dot{w}}}(k)w(k) \\
&\quad + \mathrm{i}k\boldsymbol{c}_{p_{\dot{q}}}(k)q(k)
\end{aligned}
$$

The pitch and plunge aeroelastic equations [31] are written in terms of the pitch displacement, $\alpha(k) = \theta(k)$, and plunge displacement $h(k)$, such that $q(k) = \mathrm{i}k\alpha(k)$ and $w(k) = \mathrm{i}kh(k)$. Substituting into the equation for the oscillatory pressure yields

$$
\begin{aligned}
\boldsymbol{c}_p(k) &= \mathrm{i}k\boldsymbol{c}_{p_{\dot{h}}}(k)h(k) + \boldsymbol{c}_{p_\alpha}(k)\alpha(k) + \mathrm{i}k\boldsymbol{c}_{p_{\dot{\alpha}}}(k)\alpha(k) + (\mathrm{i}k)^2\boldsymbol{c}_{p_{\ddot{h}}}(k)h(k) \\
&\quad + (\mathrm{i}k)^2\boldsymbol{c}_{p_{\ddot{\alpha}}}(k)\alpha(k)
\end{aligned}
\tag{52}
$$

where expressions for the pressure derivatives $\boldsymbol{c}_{p_{\dot{h}}}(k)$, $\boldsymbol{c}_{p_{\dot{\alpha}}}(k)$, etc. are given in [4]. The aerodynamic load and moment derivatives around the pitch axis are given by

$$
\begin{aligned}
\boldsymbol{F}_{z_\alpha}(k) &= -\boldsymbol{c}_{p_\alpha}(k) \circ \boldsymbol{s} \circ \boldsymbol{n}_z \\
\boldsymbol{M}_{y_\alpha}(k) &= -\boldsymbol{F}_{z_\alpha}(k) \circ (\boldsymbol{x}_c - x_f)
\end{aligned}
$$

and similarly for the derivatives with respect to $\dot{h}$, $\dot{\alpha}$, $\ddot{h}$, $\ddot{\alpha}$. Consequently, the generalized aerodynamic stiffness, damping and mass matrices are given by

$$
\boldsymbol{Q}_0(k_0) = \begin{pmatrix} 0 & -\sum_{i=1}^N \boldsymbol{F}_{z_\alpha}(k_0) \\ 0 & \sum_{i=1}^N \boldsymbol{M}_{y_\alpha}(k_0) \end{pmatrix}, \quad \boldsymbol{Q}_1(k_0) = \begin{pmatrix} -\sum_{i=1}^N \boldsymbol{F}_{z_{\dot{h}}}(k_0) & -\sum_{i=1}^N \boldsymbol{F}_{z_{\dot{\alpha}}}(k_0) \\ \sum_{i=1}^N \boldsymbol{M}_{y_{\dot{h}}}(k_0) & \sum_{i=1}^N \boldsymbol{M}_{y_{\dot{\alpha}}}(k_0) \end{pmatrix}
$$

$$
\boldsymbol{Q}_2(k_0) = \begin{pmatrix} -\sum_{i=1}^N \boldsymbol{F}_{z_{\ddot{h}}}(k_0) & -\sum_{i=1}^N \boldsymbol{F}_{z_{\ddot{\alpha}}}(k_0) \\ \sum_{i=1}^N \boldsymbol{M}_{y_{\ddot{h}}}(k_0) & \sum_{i=1}^N \boldsymbol{M}_{y_{\ddot{\alpha}}}(k_0) \end{pmatrix}
\tag{53}
$$

noting that the lift is defined as positive downwards in Theodorsen theory. Similarly, the steady generalized aerodynamic load vector becomes

$$
\boldsymbol{Q}_s(k_0) = \begin{pmatrix} -\sum_{i=1}^N \boldsymbol{F}_z(0) \\ \sum_{i=1}^N \boldsymbol{M}_y(0) \end{pmatrix}
\tag{54}
$$

where

$$
\begin{aligned}
\boldsymbol{F}_z(0) &= -\boldsymbol{c}_p(0) \circ \boldsymbol{s} \circ \boldsymbol{n}_z \\
\boldsymbol{M}_y(0) &= -\boldsymbol{F}_z(0) \circ (\boldsymbol{x}_c - x_f)
\end{aligned}
$$

The `PAPA` input files

- `flutter_SDPM_BSCW.py`
- `flutter_SDPM_NACA0012.py`
- `flutter_SDPM_NACA64A010.py`

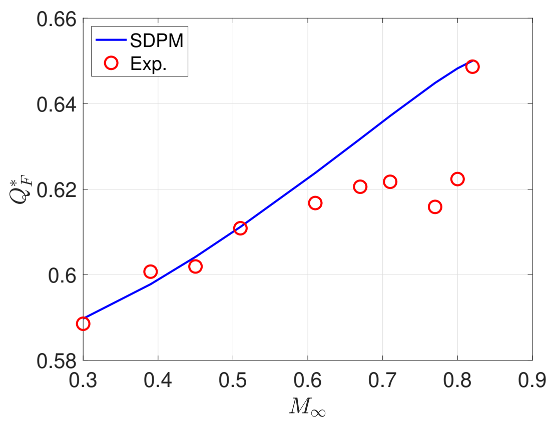
do not import or interpolate any mode shapes since the motion is described by the pitch and plunge degrees of freedom. The structural mass and stiffness matrices are calculated from

$$\boldsymbol{A} = \begin{pmatrix} m_h & S_{h\alpha} \\ S_{h\alpha} & I_\alpha \end{pmatrix}, \ \boldsymbol{E} = \begin{pmatrix} K_h & 0 \\ 0 & K_\alpha \end{pmatrix}$$

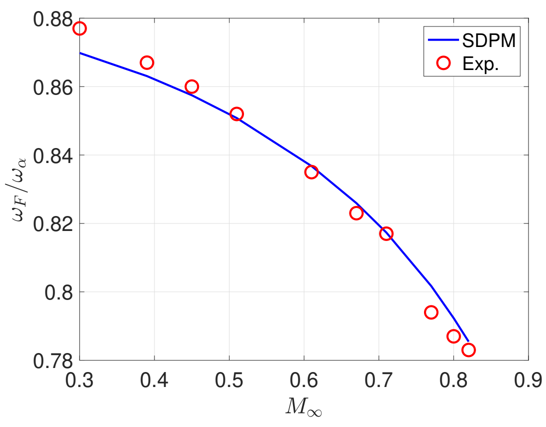
where m_h is the mass of the wing, $S_{h\alpha}$ its static imbalance around the pitch axis, I_α its moment of inertia around the pitch axis, K_h the stiffness of the plunge spring and K_α that of the pitch spring. The values of all these parameters are given in the original references, noting that $S_{h\alpha} = 0$. The rest of the `PAPA` input files are identical to those of the flexible test cases, except for

```
# Calculate flutter solution for pitch-plunge motion
Uflut,freqflut,kflut,dynpressflut,omega,zeta=
      flutsol.flutsolve_pitchplunge(body,allbodies,kvec,Uv,nmodes,
      Aphi,Bphi,Cphi,barUinf,barVinf,barWinf,c0,Mach,beta,cp_order,
      A,C,E,rho,wn,halfwing,xf0,yf0,zf0,install_dir)
```

Function `flutsol.flutsolve_pitchplunge` calculates the generalized aerodynamic load vector and matrices using equations 52 to 54. As an example, figure 14 plots the flutter speed index and flutter frequency ratio variation with Mach number for the NACA0012 test case.



(a) Flutter speed index



(b) Flutter frequency ratio

Figure 14: Variation of flutter speed index and flutter frequency ratio with Mach number for the NACA0012 test case

Again, the main difference between the DLM and SDPM analysis is the calling function:

```
# Calculate flutter solution for pitch-plunge motion
Uflut,freqflut,kflut,dynpressflut,omega,zeta=
       flutsol.flutsolve_pitchplungeDLM(body,allbodies,kvec,Uv,
       nmodes,Mach,A,C,E,rho,wn,halfwing,xf0,yf0,zf0,install_dir)
```

8 Summary

The `SDPMflut` package is a collection of functions for calculating the steady and unsteady aerodynamic loads acting on exact wing geometries and for calculating flutter solutions, if a structural model is given. It does not create any structural models, with the exception of the pitch and plunge test cases; modal models must be imported from a .mat file that has been created elsewhere. The package includes functions for creating SDPM and DLM grids for wing geometries but not for fuselages, as yet. Future versions will also include fuselage grid generation functions. The user can of course choose to use their own grids `Xp0`, `Yp0`, `Zp0`, if they have created them elsewhere. The only constraint is that the grid coordinates must be arranged in 2D arrays, so that for the SDPM that there are m chordwise panels on the lower and upper surfaces for a total of $2m$ chordwise panels, the first and last rows are the lower and upper trailing edges, respectively, while the first and last columns are the wingtips. For the DLM there are only m chordwise panels, ordered from the leading to the trailing edge. Then, the user can employ the functions in `SDPMflut` to calculate the coordinates of the control points, the components of the normal and tangential vectors, the panel areas etc., and place everything in the `body` array.

When using `SDPMflut`, the assumptions behind the source and doublet panel method must always be kept in mind:

- The flow is inviscid, irrotational and isentropic
- All deformations and displacements are small
- The flow remains attached to the surface and separated smoothly at the trailing edge
- There are no shock waves on the surface

Some of the test cases included in the package feature transonic flow with shocks on the surface; the pressure distribution around the surface for such cases cannot be modelled accurate by the SDPM or DLM. Consequently, any flutter solutions will ignore the effect of the shock waves and will fail to predict the transonic flutter dip. Future versions will include functions to correct the unsteady pressure distributions using higher fidelity steady results [6].

All drag estimates obtained from `SDPMflut` ignore viscous drag contributions. The flat plate analogy can be used to obtain an empirical estimate of steady skin friction drag, see for example [3]. The required wetted area of all the bodies in the flow can be calculated conveniently from the sums of the panel area fields, `s0`, of each element of the SDPM `body` structured array. The DLM grid does not allow wetted area calculations.

References

[1] Dimitriadis, G., *Unsteady Aerodynamics - Potential and Vortex Methods*, Wiley, 2023, https://doi.org/10.1002/9781119762560.

[2] Sánchez Martínez, M. and Dimitriadis, G., "Subsonic source and doublet panel methods," *Journal of Fluids and Structures*, Vol. 113, 2022, pp. 103624, https://doi.org/10.1016/j.jfluidstructs.2022.103624.

[3] Dimitriadis, G., Panagiotou, P., Dimopoulos, T., and Yakinthos, K., "Aerodynamic stability derivative calculations using the compressible source and doublet panel method," *Journal of Aircraft*, Vol. 61, No. 4, 2024, pp. 1034–1046, https://doi.org/10.2514/1.C037747.

[4] Dimitriadis, G., Kilimtzidis, S., Kostopoulos, V., Laraspata, V., and Soria, L., "Flutter calculations using the unsteady source and doublet panel method," *Journal of Aircraft*, Vol. 62, No. 1, 2025, pp. 117–131, https://doi.org/10.2514/1.C037891.

[5] Dimitriadis, G., Kilimtzidis, S., Kostopoulos, V., Laraspata, V., and Soria, L., "Application of the unsteady compressible source and doublet panel method to flutter calculations," *Proceedings of the International Forum on Aeroelasticity and Structural Dynamics*, No. IFASD2024-199, The Hague, Netherlands, June 2024.

[6] Dimitriadis, G., Crovato, A., Sánchez Martínez, M., Laraspata, V., Soria, L., Kilimtzidis, S., and Kostopoulos, V., "Transonic corrections for the unsteady compressible Source and Doublet Panel Method," *Journal of Aircraft*, Vol. 62, No. 3, 2025, https://doi.org/10.2514/1.C038024.

[7] Blair, M., "A compilation of the mathematics leading to the Doublet-Lattice method," Final Report WL-TR-95-3022, Air Force Wright Laboratory, 1994.

[8] Giesing, J. P., Kalman, T. P., and Rodden, W. P., "Subsonic unsteady aerodynamics for general configurations. Part I, Volume I - direct application of the nonplanar Doublet-Lattice Method," Technical Report AFFDL-TR-71-5, Air Force Flight Dynamics Laboratory, 1971.

[9] Voß, A., "An Implementation of the Vortex Lattice and the Doublet Lattice Method Version 1.05," Report DLR-IB-AE-GO-2020-137, Deutsches Zentrum für Luft- und Raumfahrt, 2020.

[10] Demasi, L., *Introduction to Unsteady Aerodynamics and Dynamic Aeroelasticity*, Springer, 2024, https://doi.org/10.1007/978-3-031-50054-1.

[11] Morino, L., "A general theory of unsteady compressible potential aerodynamics," Contractor Report CR-2464, NASA, 1974.

[12] Morino, L., Chen, L., and Suciu, E. O., "Steady and Oscillatory Subsonic and Supersonic Aerodynamics around Complex Configurations," *AIAA Journal*, Vol. 13, No. 3, 1975, pp. 368–374, https://doi.org/10.2514/3.49706.

[13] Rodden, W. P., Taylor, P. F., and McIntosh, Jr., S. C., "Further refinement of the subsonic doublet-lattice method," *Journal of Aircraft*, Vol. 35, No. 5, 1998, pp. 720–727, https://doi.org/10.2514/3.5086.

[14] Rodden, W. P. and Bellinger, E. D., "Aerodynamic lag functions, divergence, and the British flutter method," *Journal of Aircraft*, Vol. 19, No. 7, 1982, pp. 596–598, https://doi.org/10.2514/3.44772.

[15] Chen, P. C., "Damping Perturbation Method for Flutter Solution: The g-Method," *AIAA Journal*, Vol. 38, No. 9, 2000, pp. 1519–1524, https://doi.org/10.2514/2.1171.

[16] Yates, Jr, E. C., "AGARD standard aeroelastic configurations for dynamic response I – Wing 445.6," Report AGARD-R-765, AGARD, 1988.

[17] Kolbe, C. D. and Boltz, F. W., "The forces and pressure distribution at subsonic speeds of a plane wing having 45° of sweepback, an aspect ratio of 3, and a taper ratio of 0.5," Research Memorandum RM A51G31, NACA, 1951.

[18] Riley, D. R., Bird, J. D., and Fisher, L. R., "Experimental determination of the aerodynamic derivatives arising from acceleration in sideslip for a triangular, a swept, and an unswept wing," Research Memorandum RM L55A07, NACA, 1955.

[19] Queijo, M. J., Fletcher, H. S., Marple, C. G., and Hughes, F. M., "Preliminary measurements of the aerodynamic yawing derivatives of a triangular, a swept, and an unswept wing performing pure yawing oscillations, with a description of the instrumentation employed," Research Memorandum RM L55L14, NACA, 1956.

[20] Fisher, L. R., "Experimental determination of effects of frequency and amplitude on the lateral stability derivatives for a Delta, a swept, and an unswept wing oscillating in yaw," Report 1357, NACA, 1958.

[21] Lichtenstein, J. H. and Williams, J. L., "Low speed investigation of the effects of frequency and amplitude of oscillation in sideslip on the lateral stability derivatives of 60° Delta wing, a 45° sweptback wing, and an unswept wing," Research Memorandum RM L58B26, NACA, 1958.

[22] Doggett, Jr, R. V. and Farmer, M. G., "A preliminary study of the effects of vortex diffusers (winglets) on wing flutter," Technical Memorandum NASA TM X-72799, NASA, 1975.

[23] Lessing, H. C., Troutman, J. L., and Menees, G. P., "Experimental determination of the pressure distribution on a rectangular wing oscillating in the first bending mode for Mach numbers from 0.24 to 1.30," Technical Note TN-D-344, NASA, 1960.

[24] Lockman, W. K. and Lee Seegmiller, H., "An Experimental Investigation of the Subcritical and Supercritical Flow About a Swept Semispan Wing," Technical Memorandum TM-84367, NASA, 1983.

[25] Dansberry, B. E., Durham, M. H., Turnock, D. L., Silva, W. A., and Rivera, Jr., J. A., “Physical Properties of the Benchmark Models Program Supercritical Wing,” Technical Memorandum TM 4457, NASA, 1993.

[26] Bennett, R. M., “Test Cases for a Rectangular Supercritical Wing Undergoing Pitching Oscillations,” *Verification and Validation Data for Computational Unsteady Aerodynamics*, edited by L. P. Ruiz-Calavera, No. RTO-TR-26, Research and Technology Organization North Atlantic Treaty Organization, Neuilly-sur-Seine, France, 2000.

[27] Rivera, Jr, J. A., Dansberry, B. E., Durham, M. H., Bennett, R. M., and Silva, W. A., “Pressure measurements on a rectangular wing with a NACA0012 airfoil during conventional flutter,” Technical Memorandum TM-104211, NASA, 1992.

[28] Murua, J., Martìnez, P., Climent, H., van Zyl, L., and Palacios, R., “Applications of the unsteady vortex-lattice method in aircraft aeroelasticity and flight dynamics,” *Progress in Aerospace Sciences*, Vol. 55, 2012, pp. 46–72.

[29] van Zyl, L. and Mathews, E. H., “Aeroelastic Analysis of T-Tails Using an Enhanced Doublet Lattice Method,” *Journal of Aircraft*, Vol. 48, No. 3, 2011, pp. 823–831, https://doi.org/10.2514/1.C001000.

[30] van Zyl, L., *Advanced linear methods for T-tail aeroelasticity*, Ph.D. thesis, North-West University, 2011.

[31] Theodorsen, T., “General theory of aerodynamic instability and the mechanism of flutter,” Technical Report TR-496, NACA, 1935.

[32] Desmarais, R. N., “An accurate and efficient method for evaluating the kernel of the integral equation relating pressure to normalwash in unsteady potential flow,” *23d AIAA Structures, Structural Dynamics, and Materials Conference*, No. AIAA 82-0687, New Orleans, LA, May 1982, pp. 243–255, https://doi.org/10.2514/6.1982-687.

[33] Albano, E. and Rodden, W. P., “A Doublet-Lattice Method for Calculating Lift Distributions on Oscillating Surfaces in Subsonic Flows,” *AIAA Journal*, Vol. 7, No. 2, 1969, pp. 279–285, https://doi.org/10.2514/3.5086.

A Source and Doublet Panel Method

A.1 Fourier transform of nonlinear pressure equation

In applying the Fourier transform to equation 2, all the mulitplications in the time domain become convolutions in the frequency domain. The Fourier transform can be written as

$$\begin{aligned} c_p(\omega) &= \delta(\omega) - \frac{u(\omega)*u(\omega)+v(\omega)*v(\omega)+w(\omega)*w(\omega)}{Q_\infty^2} + \frac{M_\infty^2}{Q_\infty^2}\phi_x(\omega)*\phi_x(\omega) \\ &\quad -\frac{2}{Q_\infty^2}\mathrm{i}\omega\phi(\omega) + \frac{M_\infty^2}{Q_\infty^4}(\mathrm{i}\omega\phi(\omega))*(\mathrm{i}\omega\phi(\omega)) + \frac{2M_\infty^2}{Q_\infty^3}\phi_x(\omega)*(\mathrm{i}\omega\phi(\omega)) \end{aligned} \tag{55}$$

where $\delta(\omega)$ is the Kronecker Delta function and the operator $*$ denotes convolution. For sinusoidal motion at frequency ω_0, the various terms in equation 55 can be written as vectors with three elements, their frequency components at $-\omega_0$, 0 and ω_0, such that

$$\begin{aligned} \phi(\omega) &= (\phi^*(\omega_0),\ \phi(0),\ \phi(\omega_0)) \\ \mathrm{i}\omega\phi(\omega) &= (-\mathrm{i}\omega_0\phi^*(\omega_0),\ 0,\ \mathrm{i}\omega_0\phi(\omega_0)) \\ \phi_x(\omega) &= (\phi_x^*(\omega_0),\ \phi_x(0),\ \phi_x(\omega_0)) \\ \phi_y(\omega) &= (\phi_y^*(\omega_0),\ \phi_y(0),\ \phi_y(\omega_0)) \\ \phi_z(\omega) &= (\phi_z^*(\omega_0),\ \phi_z(0),\ \phi_z(\omega_0)) \\ u(\omega) &= (u_m^*(\omega_0)+\phi_x^*(\omega_0),\ U_\infty+\phi_x(0),\ u_m(\omega_0)+\phi_x(\omega_0)) \\ v(\omega) &= (v_m^*(\omega_0)+\phi_y^*(\omega_0),\ V_\infty+\phi_y(0),\ v_m(\omega_0)+\phi_y(\omega_0)) \\ w(\omega) &= (w_m^*(\omega_0)+\phi_z^*(\omega_0),\ W_\infty+\phi_z(0),\ w_m(\omega_0)+\phi_z(\omega_0)) \end{aligned}$$

where the superscript * denotes the complex conjugate. Then,

$$\phi_x(\omega)*\phi_x(\omega) = \begin{pmatrix} \phi_x^*(\omega_0)^2 \\ 2\phi_x^*(\omega_0)\phi_x(0) \\ 2\phi_x^*(\omega_0)\phi_x(\omega_0)+\phi_x(0)^2 \\ 2\phi_x(0)\phi_x(\omega_0) \\ \phi_x(\omega_0)^2 \end{pmatrix}$$

$$(\mathrm{i}\omega\phi(\omega))*(\mathrm{i}\omega\phi(\omega)) = \begin{pmatrix} (-\mathrm{i}\omega_0\phi^*(\omega_0))^2 \\ 0 \\ 2(-\mathrm{i}\omega_0\phi^*(\omega_0))(\mathrm{i}\omega_0\phi(\omega_0)) \\ 0 \\ (\mathrm{i}\omega_0\phi(\omega_0))^2 \end{pmatrix}$$

$$\phi_x(\omega)*(\mathrm{i}\omega\phi(\omega)) = \begin{pmatrix} -\phi_x^*(\omega_0)\mathrm{i}\omega_0\phi^*(\omega_0) \\ -\phi_x(0)\mathrm{i}\omega_0\phi^*(\omega_0) \\ \phi_x^*(\omega_0)\mathrm{i}\omega_0\phi(\omega_0)-\phi_x(\omega_0)\mathrm{i}\omega_0\phi^*(\omega_0) \\ \phi_x(0)\mathrm{i}\omega_0\phi(\omega_0) \\ \phi_x(\omega_0)\mathrm{i}\omega_0\phi(\omega_0) \end{pmatrix}$$

$$u(\omega) * u(\omega) = \begin{pmatrix} u^*(\omega_0)^2 \\ 2u^*(\omega_0)u(0) \\ 2u^*(\omega_0)u(\omega_0) + u(0)^2 \\ 2u(\omega_0)u(0) \\ u(\omega_0)^2 \end{pmatrix}$$

and similar expressions for $u(\omega) * u(\omega)$, $v(\omega) * v(\omega)$. In all these convolutions, the first element of the resulting vector corresponds to frequency component $\omega = -2\omega_0$, the second to $\omega = -\omega_0$, the third to $\omega = 0$, the fourth to $\omega = \omega_0$ and the fifth the $\omega = 2\omega_0$. Consequently, the pressure coefficient at $\omega = 0$ is calculated by substituting the third element of each of the convolution vector into equation 55, i.e.

$$\begin{aligned} c_p(0) &= 1 - \frac{2u^*(\omega_0)u(\omega_0) + u(0)^2}{Q_\infty^2} - \frac{2v^*(\omega_0)v(\omega_0) + v(0)^2}{Q_\infty^2} - \frac{2w^*(\omega_0)w(\omega_0) + w(0)^2}{Q_\infty^2} \\ &\quad + \frac{M_\infty^2}{Q_\infty^2}\left(2\phi_x^*(\omega_0)\phi_x(\omega_0) + \phi_x(0)^2\right) + \frac{2M_\infty^2}{Q_\infty^4}\omega_0^2\phi^*(\omega_0)\phi(\omega_0) \\ &\quad + \frac{2M_\infty^2}{Q_\infty^3}\mathrm{i}\omega_0\left(\phi_x^*(\omega_0)\phi(\omega_0) - \phi_x(\omega_0)\phi^*(\omega_0)\right) \end{aligned} \tag{56}$$

The pressure coefficient at the oscillation frequency, $\omega = \omega_0$, is obtained when using the fourth element of each of the convolution vectors, such that

$$\begin{aligned} c_p(\omega_0) &= -2\frac{u(\omega_0)u(0) + v(\omega_0)v(0) + w(\omega_0)w(0)}{Q_\infty^2} + \frac{2M_\infty^2}{Q_\infty^2}\phi_x(0)\phi_x(\omega_0) \\ &\quad - \frac{2}{Q_\infty^2}\mathrm{i}\omega_0\phi(\omega_0) + \frac{2M_\infty^2}{Q_\infty^3}\mathrm{i}\omega_0\phi_x(0)\phi(\omega_0) \end{aligned} \tag{57}$$

Finally, the pressure coefficient at twice the oscillation frequency, $\omega = 2\omega_0$, is obtained when using the fifth element of each convolution vector,

$$\begin{aligned} c_p(2\omega_0) &= -\frac{u(\omega_0)^2 + v(\omega_0)^2 + w(\omega_0)^2}{Q_\infty^2} + \frac{M_\infty^2}{Q_\infty^2}\phi_x(\omega_0)^2 \\ &\quad - \frac{M_\infty^2}{Q_\infty^4}\omega_0^2\phi(\omega_0)^2 + \frac{2M_\infty^2}{Q_\infty^3}\mathrm{i}\omega_0\phi_x(\omega_0)\phi(\omega_0) \end{aligned} \tag{58}$$

Equations 56 and 58 are nonlinear in the oscillatory potential and velocities and will not be used in the present work. In particular, equation 56 gives the zero frequency component of an oscillatory pressure, in other words the mean pressure. The steady pressure in the absence of motion is obtained by setting $\omega = 0$ directly in equation 55, such that

$$c_{p_0} = 1 - \frac{u(0)^2 + v(0)^2 + w(0)^2}{Q_\infty^2} + \frac{M_\infty^2}{Q_\infty^2}\phi_x(0)^2 \tag{59}$$

Equation 57 for $c_p(\omega_0)$ is linear in the oscillatory potential and velocities and can therefore be used for linear flutter analysis. Substituting for ω_0 from

$$\omega_0 = \frac{2Q_\infty k}{c_0}$$

leads to

$$\begin{aligned}
c_p(k) &= -2\frac{u(k)u(0)+v(k)v(0)+w(k)w(0)}{Q_\infty^2} + \frac{2M_\infty^2}{Q_\infty^2}\phi_x(0)\phi_x(k) \\
&\quad -\frac{4\mathrm{i}k}{c_0 Q_\infty}\phi(k) + \frac{4\mathrm{i}kM_\infty^2}{c_0 Q_\infty^2}\phi_x(0)\phi(k)
\end{aligned}$$

Writing out this latest expression for all the control points of all the panels on the surface leads to the vector expression

$$\begin{aligned}
\boldsymbol{c}_p(k) &= -2\frac{\boldsymbol{u}(k)\circ\boldsymbol{u}(0)+\boldsymbol{v}(k)\circ\boldsymbol{v}(0)+\boldsymbol{w}(k)\circ\boldsymbol{w}(0)}{Q_\infty^2} + \frac{2M_\infty^2}{Q_\infty^2}\boldsymbol{\phi}_x(0)\circ\boldsymbol{\phi}_x(k) \\
&\quad -\frac{4\mathrm{i}k}{c_0 Q_\infty}\boldsymbol{\phi}(k) + \frac{4\mathrm{i}kM_\infty^2}{c_0 Q_\infty^2}\boldsymbol{\phi}_x(0)\circ\boldsymbol{\phi}(k) \qquad (60)
\end{aligned}$$

We can also define the normalized potential and velocities $\bar{\boldsymbol{\phi}}(k)=\boldsymbol{\phi}(k)/Q_\infty$, $\bar{\boldsymbol{u}}(k)=\boldsymbol{u}(k)/Q_\infty$, $\bar{\boldsymbol{v}}(k)=\boldsymbol{v}(k)/Q_\infty$, $\bar{\boldsymbol{w}}(k)=\boldsymbol{w}(k)/Q_\infty$, $\bar{\boldsymbol{u}}_m(k)=\boldsymbol{u}_m(k)/Q_\infty$, $\bar{\boldsymbol{v}}_m(k)=\boldsymbol{v}_m(k)/Q_\infty$, $\bar{\boldsymbol{w}}_m(k)=\boldsymbol{w}_m(k)/Q_\infty$, $\bar{\boldsymbol{\phi}}_x(k)=\boldsymbol{\phi}_x(k)/Q_\infty$, $\bar{\boldsymbol{\phi}}_y(k)=\boldsymbol{\phi}_y(k)/Q_\infty$, $\bar{\boldsymbol{\phi}}_z(k)=\boldsymbol{\phi}_z(k)/Q_\infty$, $\bar{U}_\infty=U_\infty/Q_\infty$, $\bar{V}_\infty=V_\infty/Q_\infty$, $\bar{W}_\infty=W_\infty/Q_\infty$, such that

$$\begin{aligned}
\boldsymbol{c}_p(k) &= -2\left(\bar{\boldsymbol{u}}(k)\circ\bar{\boldsymbol{u}}(0)+\bar{\boldsymbol{v}}(k)\circ\bar{\boldsymbol{v}}(0)+\bar{\boldsymbol{w}}(k)\circ\bar{\boldsymbol{w}}(0)\right)+2M_\infty^2\bar{\boldsymbol{\phi}}_x(0)\circ\bar{\boldsymbol{\phi}}_x(k) \\
&\quad -\frac{4\mathrm{i}k}{c_0}\bar{\boldsymbol{\phi}}(k)+\frac{4\mathrm{i}kM_\infty^2}{c_0}\bar{\boldsymbol{\phi}}_x(0)\circ\bar{\boldsymbol{\phi}}(k) \qquad (61)
\end{aligned}$$

In order to proceed further, we note that the normalized versions of equations 9, 10 and 11 become

$$\bar{\boldsymbol{\mu}}_n(k) = -\left(\frac{1}{\beta}\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi+\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta+\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta\right) \qquad (62)$$

$$\bar{\boldsymbol{\phi}}(k) = -\boldsymbol{K}(k)\bar{\boldsymbol{\mu}}_n(k) \qquad (63)$$

$$\bar{\boldsymbol{\phi}}_x(k)=\boldsymbol{K}_x(k)\bar{\boldsymbol{\mu}}_n(k),\ \bar{\boldsymbol{\phi}}_y(k)=\boldsymbol{K}_y(k)\bar{\boldsymbol{\mu}}_n(k),\ \bar{\boldsymbol{\phi}}_z(k)=\boldsymbol{K}_z(k)\bar{\boldsymbol{\mu}}_n(k) \qquad (64)$$

$$\bar{\boldsymbol{u}}(k)=\bar{\boldsymbol{u}}_m(k)+\bar{\boldsymbol{\phi}}_x(k),\ \bar{\boldsymbol{v}}(k)=\bar{\boldsymbol{v}}_m(k)+\bar{\boldsymbol{\phi}}_y(k),\ \bar{\boldsymbol{w}}(k)=\bar{\boldsymbol{w}}_m(k)+\bar{\boldsymbol{\phi}}_z(k) \qquad (65)$$

Let us treat the terms in equation 61 one by one. Substituting from expressions 62 to 65, the term $\bar{\boldsymbol{u}}(k)\circ\bar{\boldsymbol{u}}(0)$ becomes

$$\begin{aligned}
\bar{\boldsymbol{u}}(k)\circ\bar{\boldsymbol{u}}(0) &= (\bar{\boldsymbol{u}}_m(k)+\bar{\boldsymbol{\phi}}_x(k))\circ\bar{\boldsymbol{u}}(0)=(\bar{\boldsymbol{u}}_m(k)+\boldsymbol{K}_x(k)\bar{\boldsymbol{\mu}}_n(k))\circ\bar{\boldsymbol{u}}(0) \\
&= \bar{\boldsymbol{u}}_m(k)\circ\bar{\boldsymbol{u}}(0)-\left(\boldsymbol{K}_x(k)\left(\frac{1}{\beta}\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi+\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta+\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta\right)\right)\circ\bar{\boldsymbol{u}}(0) \\
&= \bar{\boldsymbol{u}}_m(k)\circ\bar{\boldsymbol{u}}(0)-\frac{1}{\beta}\boldsymbol{K}_x(k)\circ\bar{\boldsymbol{u}}(0)(\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi)-\boldsymbol{K}_x(k)\circ\bar{\boldsymbol{u}}(0)(\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta) \\
&\quad -\boldsymbol{K}_x(k)\circ\bar{\boldsymbol{u}}(0)(\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta) \qquad (66)
\end{aligned}$$

Similarly,

$$\bar{\boldsymbol{v}}(k)\circ\bar{\boldsymbol{v}}(0) = \bar{\boldsymbol{v}}_m(k)\circ\bar{\boldsymbol{v}}(0)-\frac{1}{\beta}\boldsymbol{K}_y(k)\circ\bar{\boldsymbol{v}}(0)(\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi)-\boldsymbol{K}_y(k)\circ\bar{\boldsymbol{v}}(0)(\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta)$$

$$\begin{aligned} & -\boldsymbol{K}_y(k)\circ\bar{\boldsymbol{v}}(0)(\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta) && (67)\\ \bar{\boldsymbol{w}}(k)\circ\bar{\boldsymbol{w}}(0) &= \bar{\boldsymbol{w}}_m(k)\circ\bar{\boldsymbol{w}}(0)-\frac{1}{\beta}\boldsymbol{K}_z(k)\circ\bar{\boldsymbol{w}}(0)(\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi)-\boldsymbol{K}_z(k)\circ\bar{\boldsymbol{w}}(0)(\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta)\\ & -\boldsymbol{K}_z(k)\circ\bar{\boldsymbol{w}}(0)(\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta) && (68)\end{aligned}$$

Furthermore,

$$\begin{aligned}\bar{\boldsymbol{\phi}}_x(0)\circ\bar{\boldsymbol{\phi}}_x(k) &= \bar{\boldsymbol{\phi}}_x(0)\circ(\boldsymbol{K}_x(k)\bar{\boldsymbol{\mu}}_n(k))\\ &= -\boldsymbol{K}_x(k)\circ\bar{\boldsymbol{\phi}}_x(0)\left(\frac{1}{\beta}\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi+\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta+\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta\right)\\ &= -\frac{1}{\beta}\boldsymbol{K}_x(k)\circ\bar{\boldsymbol{\phi}}_x(0)(\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi)-\boldsymbol{K}_x(k)\circ\bar{\boldsymbol{\phi}}_x(0)(\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta)\\ & -\boldsymbol{K}_x(k)\circ\bar{\boldsymbol{\phi}}_x(0)(\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta) && (69)\end{aligned}$$

while

$$\begin{aligned}\bar{\boldsymbol{\phi}}(k) &= -\boldsymbol{K}(k)\bar{\boldsymbol{\mu}}_n(k)\\ &= \boldsymbol{K}(k)\left(\frac{1}{\beta}\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi+\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta+\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta\right)\\ &= \frac{1}{\beta}\boldsymbol{K}(k)(\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi)+\boldsymbol{K}(k)(\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta)\\ & +\boldsymbol{K}(k)(\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta) && (70)\end{aligned}$$

and, finally,

$$\begin{aligned}\bar{\boldsymbol{\phi}}_x(0)\circ\bar{\boldsymbol{\phi}}(k) &= \bar{\boldsymbol{\phi}}_x(0)\circ(\boldsymbol{K}(k)\bar{\boldsymbol{\mu}}_n(k))\\ &= \boldsymbol{K}(k)\circ\bar{\boldsymbol{\phi}}_x(0)\left(\frac{1}{\beta}\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi+\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta+\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta\right)\\ &= \frac{1}{\beta}\boldsymbol{K}(k)\circ\bar{\boldsymbol{\phi}}_x(0)(\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi)+\boldsymbol{K}(k)\circ\bar{\boldsymbol{\phi}}_x(0)(\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta)\\ & +\boldsymbol{K}(k)\circ\bar{\boldsymbol{\phi}}_x(0)(\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta) && (71)\end{aligned}$$

Substituting equations 66 to 71 into equation 61 leads to

$$\begin{aligned}\boldsymbol{c}_p(k) &= -2\bar{\boldsymbol{u}}_m(k)\circ\bar{\boldsymbol{u}}(0)+\frac{2}{\beta}\boldsymbol{K}_x(k)\circ\bar{\boldsymbol{u}}(0)(\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi)+2\boldsymbol{K}_x(k)\circ\bar{\boldsymbol{u}}(0)(\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta)\\ & +2\boldsymbol{K}_x(k)\circ\bar{\boldsymbol{u}}(0)(\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta)-2\bar{\boldsymbol{v}}_m(k)\circ\bar{\boldsymbol{v}}(0)+\frac{2}{\beta}\boldsymbol{K}_y(k)\circ\bar{\boldsymbol{v}}(0)(\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi)\\ & +2\boldsymbol{K}_y(k)\circ\bar{\boldsymbol{v}}(0)(\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta)+2\boldsymbol{K}_y(k)\circ\bar{\boldsymbol{v}}(0)(\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta)-2\bar{\boldsymbol{w}}_m(k)\circ\bar{\boldsymbol{w}}(0)\\ & +\frac{2}{\beta}\boldsymbol{K}_z(k)\circ\bar{\boldsymbol{w}}(0)(\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi)+2\boldsymbol{K}_z(k)\circ\bar{\boldsymbol{w}}(0)(\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta)\\ & +2\boldsymbol{K}_z(k)\circ\bar{\boldsymbol{w}}(0)(\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta)-\frac{2M_\infty^2}{\beta}\boldsymbol{K}_x(k)\circ\bar{\boldsymbol{\phi}}_x(0)(\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi)\\ & -2M_\infty^2\boldsymbol{K}_x(k)\circ\bar{\boldsymbol{\phi}}_x(0)(\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta)-2M_\infty^2\boldsymbol{K}_x(k)\circ\bar{\boldsymbol{\phi}}_x(0)(\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta)\end{aligned}$$

$$
-\frac{4\mathrm{i}k}{c_0\beta}\boldsymbol{K}(k)(\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi)-\frac{4\mathrm{i}k}{c_0}\boldsymbol{K}(k)(\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta)-\frac{4\mathrm{i}k}{c_0}\boldsymbol{K}(k)(\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta)
+\frac{4\mathrm{i}kM_\infty^2}{c_0\beta}\boldsymbol{K}(k)\circ\bar{\boldsymbol{\phi}}_x(0)(\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi)+\frac{4\mathrm{i}kM_\infty^2}{c_0}\boldsymbol{K}(k)\circ\bar{\boldsymbol{\phi}}_x(0)(\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta)
+\frac{4\mathrm{i}kM_\infty^2}{c_0}\boldsymbol{K}(k)\circ\bar{\boldsymbol{\phi}}_x(0)(\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta)
$$

This latest expression can be tidied up by grouping the terms in $(\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi)$, $(\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta)$ and $(\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta)$ with the help of the coefficients

$$
\begin{aligned}
\bar{\boldsymbol{C}}_0(k) &= -2\boldsymbol{K}_x(k)\circ\bar{\boldsymbol{u}}(0)-2\boldsymbol{K}_y(k)\circ\bar{\boldsymbol{v}}(0)-2\boldsymbol{K}_z(k)\circ\bar{\boldsymbol{w}}(0)+2M_\infty^2\boldsymbol{K}_x(k)\circ\bar{\boldsymbol{\phi}}_x(0)\\
\bar{\boldsymbol{C}}_1(k) &= 2\boldsymbol{K}(k)-2M_\infty^2\boldsymbol{K}(k)\circ\bar{\boldsymbol{\phi}}_x(0)
\end{aligned}
\tag{72}
$$

Consequently, the final expression for the oscillatory pressure coefficient on the surface becomes

$$
\begin{aligned}
\boldsymbol{c}_p(k) = &-\frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\xi\circ\bar{\boldsymbol{u}}_m(k))-2\bar{\boldsymbol{u}}(0)\circ\bar{\boldsymbol{u}}_m(k)-\bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\eta\circ\bar{\boldsymbol{v}}_m(k))-2\bar{\boldsymbol{v}}(0)\circ\bar{\boldsymbol{v}}_m(k)\\
&-\bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\zeta\circ\bar{\boldsymbol{w}}_m(k))-2\bar{\boldsymbol{w}}(0)\circ\bar{\boldsymbol{w}}_m(k)-\frac{2\mathrm{i}k}{c_0\beta}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\xi\circ\bar{\boldsymbol{u}}_m(k))\\
&-\frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\eta\circ\bar{\boldsymbol{v}}_m(k))-\frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\zeta\circ\bar{\boldsymbol{w}}_m(k))
\end{aligned}
\tag{73}
$$

A.1.1 Linearized pressure on the surface

The first order oscillatory pressure equation is obtained by neglecting all nonlinear terms in equation 2, such that

$$
c_p(\omega)=-\frac{2\phi_x(\omega)}{Q_\infty}-\frac{2}{Q_\infty^2}\mathrm{i}\omega\phi(\omega)
$$

Substituting for $\omega=2kQ_\infty/c_0$ yields the linearized pressure coefficient on the panels

$$
\boldsymbol{c}_p(k)=-2\bar{\boldsymbol{\phi}}_x(k)-\frac{4\mathrm{i}k}{c_0}\bar{\boldsymbol{\phi}}(k)
\tag{74}
$$

Substituting from equations 62 to 64 in expression 74 leads to

$$
\begin{aligned}
\boldsymbol{c}_p(k) &= -2\boldsymbol{K}_x(k)\bar{\boldsymbol{\mu}}_n(k)+\frac{4\mathrm{i}k}{c_0}\boldsymbol{K}(k)\bar{\boldsymbol{\mu}}_n(k)\\
&= 2\boldsymbol{K}_x(k)\left(\frac{1}{\beta}\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi+\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta+\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta\right)\\
&\quad-\frac{4\mathrm{i}k}{c_0}\boldsymbol{K}(k)\left(\frac{1}{\beta}\bar{\boldsymbol{u}}_m(k)\circ\boldsymbol{n}_\xi+\bar{\boldsymbol{v}}_m(k)\circ\boldsymbol{n}_\eta+\bar{\boldsymbol{w}}_m(k)\circ\boldsymbol{n}_\zeta\right)
\end{aligned}
\tag{75}
$$

which is the linearized equivalent of equation 73.

A.2 Nonlinear pressure derivatives for rigid body motion

The pressure coefficient at the oscillation frequency, k, also referred to as the oscillatory pressure, is given by equation 18

$$\begin{aligned}
\boldsymbol{c}_p(k) \quad = \quad & -\frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\xi \circ \bar{\boldsymbol{u}}_m(k)) - 2\bar{\boldsymbol{u}}(0) \circ \bar{\boldsymbol{u}}_m(k) - \bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\eta \circ \bar{\boldsymbol{v}}_m(k)) - 2\bar{\boldsymbol{v}}(0) \circ \bar{\boldsymbol{v}}_m(k) \\
& -\bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\zeta \circ \bar{\boldsymbol{w}}_m(k)) - 2\bar{\boldsymbol{w}}(0) \circ \bar{\boldsymbol{w}}_m(k) - \frac{2\mathrm{i}k}{c_0\beta}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\xi \circ \bar{\boldsymbol{u}}_m(k)) \\
& -\frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\eta \circ \bar{\boldsymbol{v}}_m(k)) - \frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\zeta \circ \bar{\boldsymbol{w}}_m(k))
\end{aligned} \tag{76}$$

Rigid-body motion is defined here as the translation and rotation of the centre of gravity of a body in a body-fixed coordinate system, whereby the x axis is aligned with the chord or fuselage centreline, the y axis is aligned with the span and the z axis is perpendicular to the other two axes. To avoid confusion with the symbols used for the fluid velocities, the translation velocities are denoted by $\dot{x}(t)$, $\dot{y}(t)$ and $\dot{z}(t)$. The rotations are denoted by $\phi(t)$, $\theta(t)$, $\psi(t)$ for the roll, pitch and yaw angles. Assuming that the free stream vector is given by $\boldsymbol{Q}_\infty = (U_\infty,\ V_\infty,\ W_\infty)$ and assuming that all rotation angles are small, the relative velocities between the flow and the body are given by

$$\begin{aligned}
\boldsymbol{u}_m(t) &= V_\infty\psi(t) - W_\infty\theta(t) - \dot{\theta}(t)\boldsymbol{z}_c + \dot{\psi}(t)\boldsymbol{y}_c - \dot{x}(t) \\
\boldsymbol{v}_m(t) &= -U_\infty\psi(t) + W_\infty\phi(t) + \dot{\phi}(t)\boldsymbol{z}_c - \dot{\psi}(t)\boldsymbol{x}_c - \dot{y}(t) \\
\boldsymbol{w}_m(t) &= U_\infty\theta(t) - V_\infty\phi(t) + \dot{\theta}(t)\boldsymbol{x}_c - \dot{\phi}(t)\boldsymbol{y}_c - \dot{z}(t)
\end{aligned} \tag{77}$$

where the first two terms in each velocity component are due to the rotation of the freestream relative to the body-fixed axes, the next two terms are due to the rotational velocities of the centre of gravity and the last terms due to the translation velocities of the centre of gravity. The $N \times 1$ vectors $\boldsymbol{x}_c$, $\boldsymbol{y}_c$, $\boldsymbol{z}_c$, are the coordinates of the N panel control points, assuming that the centre of the coordinate system is the centre of gravity; consequently the relative velocity vectors $\boldsymbol{u}_m(t)$, $\boldsymbol{v}_m(t)$, $\boldsymbol{w}_m(t)$ are also $N \times 1$. Applying the Fourier transform to equations 77 leads to

$$\begin{aligned}
\boldsymbol{u}_m(\omega) &= V_\infty\psi(\omega) - W_\infty\theta(\omega) - \mathrm{i}\omega\theta(\omega)\boldsymbol{z}_c + \mathrm{i}\omega\psi(\omega)\boldsymbol{y}_c - \mathrm{i}\omega x(\omega) \\
\boldsymbol{v}_m(\omega) &= -U_\infty\psi(\omega) + W_\infty\phi(\omega) + \mathrm{i}\omega\phi(\omega)\boldsymbol{z}_c - \mathrm{i}\omega\psi(\omega)\boldsymbol{x}_c - \mathrm{i}\omega y(\omega) \\
\boldsymbol{w}_m(\omega) &= U_\infty\theta(\omega) - V_\infty\phi(\omega) + \mathrm{i}\omega\theta(\omega)\boldsymbol{x}_c - \mathrm{i}\omega\phi(\omega)\boldsymbol{y}_c - \mathrm{i}\omega z(\omega)
\end{aligned} \tag{78}$$

Recall the definition of the reduced frequency $k = \omega c_0/2Q_\infty$, such that $\omega = 2kQ_\infty/c_0$. Dividing equations 78 throughout by Q_∞ leads to

$$\begin{aligned}
\bar{\boldsymbol{u}}_m(k) &= \bar{V}_\infty\psi(k) - \bar{W}_\infty\theta(k) - \frac{2\mathrm{i}k}{c_0}\theta(k)\boldsymbol{z}_c + \frac{2\mathrm{i}k}{c_0}\psi(k)\boldsymbol{y}_c - \frac{2\mathrm{i}k}{c_0}x(k) \\
\bar{\boldsymbol{v}}_m(k) &= -\bar{U}_\infty\psi(k) + \bar{W}_\infty\phi(k) + \frac{2\mathrm{i}k}{c_0}\phi(k)\boldsymbol{z}_c - \frac{2\mathrm{i}k}{c_0}\psi(k)\boldsymbol{x}_c - \frac{2\mathrm{i}k}{c_0}y(k) \\
\bar{\boldsymbol{w}}_m(k) &= \bar{U}_\infty\theta(k) - \bar{V}_\infty\phi(k) + \frac{2\mathrm{i}k}{c_0}\theta(k)\boldsymbol{x}_c - \frac{2\mathrm{i}k}{c_0}\phi(k)\boldsymbol{y}_c - \frac{2\mathrm{i}k}{c_0}z(k)
\end{aligned} \tag{79}$$

Aircraft aerodynamic stability derivatives are usually expressed as derivatives of non-dimensional aerodynamic load coefficients with respect to non-dimensional quantities

$$\bar{u}(t)=\frac{1}{Q_\infty}\dot{x}(t),\ \bar{v}(t)=\frac{1}{Q_\infty}\dot{y}(t),\ \bar{w}(t)=\frac{1}{Q_\infty}\dot{z}(t)$$

$$\bar{\dot{u}}(t)=\frac{c_0}{2Q_\infty^2}\ddot{x}(t),\ \bar{\dot{v}}(t)=\frac{b}{2Q_\infty^2}\ddot{y}(t),\ \bar{\dot{w}}(t)=\frac{c_0}{2Q_\infty^2}\ddot{z}(t)$$

$$\bar{p}(t)=\frac{b}{2Q_\infty}\dot{\phi}(t),\ \bar{q}(t)=\frac{c_0}{2Q_\infty}\dot{\theta}(t),\ \bar{r}(t)=\frac{b}{2Q_\infty}\dot{\psi}(t)$$

$$\bar{\dot{p}}(t)=\left(\frac{b}{2Q_\infty}\right)^2\ddot{\phi}(t),\ \bar{\dot{q}}(t)=\left(\frac{c_0}{2Q_\infty}\right)^2\ddot{\theta}(t),\ \bar{\dot{r}}(t)=\left(\frac{b}{2Q_\infty}\right)^2\ddot{\psi}(t)$$

where b is the span. In the frequency domain, these definitions become

$$\begin{aligned}
\bar{u}(k) &= \left(\frac{2\mathrm{i}k}{c_0}\right)x(k),\ \bar{v}(k)=\left(\frac{2\mathrm{i}k}{c_0}\right)y(k),\ \bar{w}(k)=\left(\frac{2\mathrm{i}k}{c_0}\right)z(k)\\
\bar{\dot{u}}(k) &= \frac{c_0}{2}\left(\frac{2\mathrm{i}k}{c_0}\right)^2x(k),\ \bar{\dot{v}}(k)=\frac{b}{2}\left(\frac{2\mathrm{i}k}{c_0}\right)^2y(k),\ \bar{\dot{w}}(k)=\frac{c_0}{2}\left(\frac{2\mathrm{i}k}{c_0}\right)^2z(k)\\
\bar{p}(k) &= \frac{b}{2}\left(\frac{2\mathrm{i}k}{c_0}\right)\phi(k),\ \bar{q}(k)=\frac{c_0}{2}\left(\frac{2\mathrm{i}k}{c_0}\right)\theta(k),\ \bar{r}(k)=\frac{b}{2}\left(\frac{2\mathrm{i}k}{c_0}\right)\psi(k) \qquad (80)\\
\bar{\dot{p}}(k) &= \left(\frac{b}{2}\right)^2\left(\frac{2\mathrm{i}k}{c_0}\right)^2\phi(k),\ \bar{\dot{q}}(k)=\left(\frac{c_0}{2}\right)^2\left(\frac{2\mathrm{i}k}{c_0}\right)^2\theta(k),\ \bar{\dot{r}}(k)=\left(\frac{b}{2}\right)^2\left(\frac{2\mathrm{i}k}{c_0}\right)^2\psi(k)
\end{aligned}$$

In order to obtain the derivatives of $\boldsymbol{c}_p(k)$ with respect to $\bar{u}(k)$ and $\bar{\dot{u}}(k)$, equations 79 are substituted into equation 76, after setting $\bar{y}(k)=\bar{z}(k)=\phi(k)=\theta(k)=\psi(k)=0$. Then, equation 76 becomes

$$\begin{aligned}
\boldsymbol{c}_p(k) &= -\frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\xi\left(-\frac{2\mathrm{i}k}{c_0}x(k)\right)\right)-2\bar{\boldsymbol{u}}(0)\left(-\frac{2\mathrm{i}k}{c_0}x(k)\right)\\
&\quad -\frac{2\mathrm{i}k}{c_0\beta}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\xi\left(-\frac{2\mathrm{i}k}{c_0}x(k)\right)\right)
\end{aligned}$$

where the Hadamard product is no longer required since $x(k)$ is a scalar. Substituting for $\bar{u}(k)$ and $\bar{\dot{u}}(k)$ from equations 80 leads to

$$\boldsymbol{c}_p(k) = \frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)\boldsymbol{n}_\xi\bar{u}(k)+2\bar{\boldsymbol{u}}(0)\bar{u}(k)+\frac{2}{c_0\beta}\bar{\boldsymbol{C}}_1(k)\boldsymbol{n}_\xi\bar{\dot{u}}(k)$$

Consequently, the pressure derivatives with respect to $\bar{u}(k)$ and $\bar{\dot{u}}(k)$ are given by

$$\boldsymbol{c}_{p_u}(k) = \frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)\boldsymbol{n}_\xi+2\bar{\boldsymbol{u}}(0) \qquad (81)$$

$$\boldsymbol{c}_{p_{\dot{u}}}(k) = \frac{2}{c_0\beta}\bar{\boldsymbol{C}}_1(k)\boldsymbol{n}_\xi \qquad (82)$$

Similarly, for the derivatives of $\boldsymbol{c}_p(k)$ with respect to $\bar{v}(k)$ and $\bar{\dot{v}}(k)$, we set $x(k) = z(k) = \phi(k) = \theta(k) = \psi(k) = 0$ in equations 79 and substitute them in equation 76 to obtain

$$\begin{aligned}\boldsymbol{c}_p(k) &= -\bar{\boldsymbol{C}}_0(k)\boldsymbol{n}_\eta\left(-\frac{2\mathrm{i}k}{c_0}y(k)\right) - 2\bar{\boldsymbol{v}}(0)\left(-\frac{2\mathrm{i}k}{c_0}y(k)\right)\\ &\quad -\frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)\boldsymbol{n}_\eta\left(-\frac{2\mathrm{i}k}{c_0}y(k)\right)\end{aligned}$$

Substituting for $\bar{v}(k)$ and $\bar{\dot{v}}(k)$ from equations 80 leads to the pressure derivatives

$$\boldsymbol{c}_{p_v}(k) = \bar{\boldsymbol{C}}_0(k)\boldsymbol{n}_\eta + 2\bar{\boldsymbol{v}}(0) \tag{83}$$

$$\boldsymbol{c}_{p_{\dot{v}}}(k) = \frac{2}{b}\bar{\boldsymbol{C}}_1(k)\boldsymbol{n}_\eta \tag{84}$$

Repeating the procedure for the derivatives of $\boldsymbol{c}_p(k)$ with respect to $\bar{w}$ and $\bar{\dot{w}}$, we obtain

$$\boldsymbol{c}_{p_w}(k) = \bar{\boldsymbol{C}}_0(k)\boldsymbol{n}_\zeta + 2\bar{\boldsymbol{w}}(0) \tag{85}$$

$$\boldsymbol{c}_{p_{\dot{w}}}(k) = \frac{2}{c_0}\bar{\boldsymbol{C}}_1(k)\boldsymbol{n}_\zeta \tag{86}$$

The derivatives with respect to $\phi(k)$, $\bar{p}(k)$ and $\bar{\dot{p}}(k)$ are obtained by setting $x(k) = y(k) = z(k) = \theta(k) = \psi(k) = 0$ in equations 79 and substituting them into equation 76, so that

$$\begin{aligned}\boldsymbol{c}_p(k) &= -\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\eta \circ \left(\bar{W}_\infty\phi(k) + \frac{2\mathrm{i}k}{c_0}\phi(k)\boldsymbol{z}_c\right)\right) - 2\bar{\boldsymbol{v}}(0)\circ\left(\bar{W}_\infty\phi(k) + \frac{2\mathrm{i}k}{c_0}\phi(k)\boldsymbol{z}_c\right)\\ &\quad -\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\zeta \circ \left(-\bar{V}_\infty\phi(k) - \frac{2\mathrm{i}k}{c_0}\phi(k)\boldsymbol{y}_c\right)\right) - 2\bar{\boldsymbol{w}}(0)\circ\left(-\bar{V}_\infty\phi(k) - \frac{2\mathrm{i}k}{c_0}\phi(k)\boldsymbol{y}_c\right)\\ &\quad -\frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\eta \circ \left(\bar{W}_\infty\phi(k) + \frac{2\mathrm{i}k}{c_0}\phi(k)\boldsymbol{z}_c\right)\right)\\ &\quad -\frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\zeta \circ \left(-\bar{V}_\infty\phi(k) - \frac{2\mathrm{i}k}{c_0}\phi(k)\boldsymbol{y}_c\right)\right)\end{aligned}$$

Substituting from equations 80 for $\bar{p}(k)$ and $\bar{\dot{p}}(k)$ results in the derivatives

$$\boldsymbol{c}_{p_\phi}(k) = -\bar{\boldsymbol{C}}_0(k)\boldsymbol{n}_\eta\bar{W}_\infty - 2\bar{\boldsymbol{v}}(0)\bar{W}_\infty + \bar{\boldsymbol{C}}_0(k)\boldsymbol{n}_\zeta\bar{V}_\infty + 2\bar{\boldsymbol{w}}(0)\bar{V}_\infty \tag{87}$$

$$\begin{aligned}\boldsymbol{c}_{p_p}(k) &= \frac{2}{b}\left(-\bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\eta \circ \boldsymbol{z}_c) - 2\bar{\boldsymbol{v}}(0)\circ\boldsymbol{z}_c + \bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\zeta \circ \boldsymbol{y}_c) + 2\bar{\boldsymbol{w}}(0)\boldsymbol{y}_c\right.\\ &\quad \left.-\bar{\boldsymbol{C}}_1(k)\boldsymbol{n}_\eta\bar{W}_\infty + \bar{\boldsymbol{C}}_1(k)\boldsymbol{n}_\zeta\bar{V}_\infty\right)\end{aligned} \tag{88}$$

$$\boldsymbol{c}_{p_{\dot{p}}}(k) = \left(\frac{2}{b}\right)^2\left(-\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\eta \circ \boldsymbol{z}_c) + \bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\zeta \circ \boldsymbol{y}_c)\right) \tag{89}$$

Setting $x(k) = y(k) = z(k) = \phi(k) = \psi(k) = 0$ in equations 79 and substituting them into equation 76 leads to

$$\boldsymbol{c}_p(k) = -\frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\xi \circ \left(-\bar{W}_\infty\theta(k) - \frac{2\mathrm{i}k}{c_0}\theta(k)\boldsymbol{z}_c\right)\right) - 2\bar{\boldsymbol{u}}(0)\circ\left(-\bar{W}_\infty\theta(k) - \frac{2\mathrm{i}k}{c_0}\theta(k)\boldsymbol{z}_c\right)$$

$$
\begin{aligned}
&-\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\zeta \circ \left(\bar{U}_\infty \theta(k) + \frac{2\mathrm{i}k}{c_0}\theta(k)\boldsymbol{x}_c\right)\right) - 2\bar{\boldsymbol{w}}(0) \circ \left(\bar{U}_\infty \theta(k) + \frac{2\mathrm{i}k}{c_0}\theta(k)\boldsymbol{x}_c\right) \\
&-\frac{2\mathrm{i}k}{c_0\beta}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\xi \circ \left(-\bar{W}_\infty \theta(k) - \frac{2\mathrm{i}k}{c_0}\theta(k)\boldsymbol{z}_c\right)\right) \\
&-\frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\zeta \circ \left(\bar{U}_\infty \theta(k) + \frac{2\mathrm{i}k}{c_0}\theta(k)\boldsymbol{x}_c\right)\right)
\end{aligned}
$$

Substituting from equations 80 for $\bar{q}(k)$ and $\bar{\dot{q}}(k)$ leads to

$$
\begin{aligned}
\boldsymbol{c}_{p_\theta}(k) &= \frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)\boldsymbol{n}_\xi \bar{W}_\infty + 2\bar{\boldsymbol{u}}(0)\bar{W}_\infty - \bar{\boldsymbol{C}}_0(k)\boldsymbol{n}_\zeta \bar{U}_\infty - 2\bar{\boldsymbol{w}}(0)\bar{U}_\infty && (90) \\
\boldsymbol{c}_{p_q}(k) &= \frac{2}{c_0}\left(\frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\xi \circ \boldsymbol{z}_c) + 2\bar{\boldsymbol{u}}(0) \circ \boldsymbol{z}_c - \bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\zeta \circ \boldsymbol{x}_c) - 2\bar{\boldsymbol{w}}(0) \circ \boldsymbol{x}_c\right. \\
&\quad \left. + \frac{1}{\beta}\bar{\boldsymbol{C}}_1(k)\boldsymbol{n}_\xi \bar{W}_\infty - \bar{\boldsymbol{C}}_1(k)\boldsymbol{n}_\zeta \bar{U}_\infty\right) && (91) \\
\boldsymbol{c}_{p_{\dot{q}}}(k) &= \left(\frac{2}{c_0}\right)^2 \left(\frac{1}{\beta}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\xi \circ \boldsymbol{z}_c) - \bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\zeta \circ \boldsymbol{x}_c)\right) && (92)
\end{aligned}
$$

Finally, setting $x(k) = y(k) = z(k) = \phi(k) = \theta(k) = 0$ in equations 79 and substituting them into equation 76 leads to

$$
\begin{aligned}
\boldsymbol{c}_p(k) = &-\frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\xi \circ \left(\bar{V}_\infty \psi(k) + \frac{2\mathrm{i}k}{c_0}\psi(k)\boldsymbol{y}_c\right)\right) - 2\bar{\boldsymbol{u}}(0) \circ \left(\bar{V}_\infty \psi(k) + \frac{2\mathrm{i}k}{c_0}\psi(k)\boldsymbol{y}_c\right) \\
&-\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\eta \circ \left(-\bar{U}_\infty \psi(k) - \frac{2\mathrm{i}k}{c_0}\psi(k)\boldsymbol{x}_c\right)\right) - 2\bar{\boldsymbol{v}}(0) \circ \left(-\bar{U}_\infty \psi(k) - \frac{2\mathrm{i}k}{c_0}\psi(k)\boldsymbol{x}_c\right) \\
&-\frac{2\mathrm{i}k}{c_0\beta}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\xi \circ \left(\bar{V}_\infty \psi(k) + \frac{2\mathrm{i}k}{c_0}\psi(k)\boldsymbol{y}_c\right)\right) \\
&-\frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\eta \circ \left(-\bar{U}_\infty \psi(k) - \frac{2\mathrm{i}k}{c_0}\psi(k)\boldsymbol{x}_c\right)\right)
\end{aligned}
$$

and the pressure derivatives in the yaw direction become

$$
\begin{aligned}
\boldsymbol{c}_{p_\psi}(k) &= -\frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)\boldsymbol{n}_\xi \bar{V}_\infty - 2\bar{\boldsymbol{u}}(0)\bar{V}_\infty + \bar{\boldsymbol{C}}_0(k)\boldsymbol{n}_\eta \bar{U}_\infty + 2\bar{\boldsymbol{v}}(0)\bar{U}_\infty && (93) \\
\boldsymbol{c}_{p_r}(k) &= \frac{2}{b}\left(-\frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\xi \circ \boldsymbol{y}_c) - 2\bar{\boldsymbol{u}}(0) \circ \boldsymbol{y}_c + \bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\eta \circ \boldsymbol{x}_c) + 2\bar{\boldsymbol{v}}(0) \circ \boldsymbol{x}_c\right. \\
&\quad \left. - \frac{1}{\beta}\bar{\boldsymbol{C}}_1(k)\boldsymbol{n}_\xi \bar{V}_\infty + \bar{\boldsymbol{C}}_1(k)\boldsymbol{n}_\eta \bar{U}_\infty\right) && (94) \\
\boldsymbol{c}_{p_{\dot{r}}}(k) &= \left(\frac{2}{b}\right)^2 \left(-\frac{1}{\beta}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\xi \circ \boldsymbol{y}_c) + \bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\eta \circ \boldsymbol{x}_c)\right) && (95)
\end{aligned}
$$

It should be noted that $c_0/2$ in expressions 81 to 95 can be replaced by any other reference chordwise length. Similarly, $b/2$ can be replaced by any other reference spanwise length.

A.3 Pressure derivatives for pitching and plunging motion

Pitching and plunging motion is a special guess of rigid body motion whereby only two degrees of freedom, pitch $\alpha(t)$ and plunge $h(t)$ displacements; the latter is defined positive downwards. The coordinate system is usually not centred on the pitch axis, so that the motion-induced velocities are given by a modified version of equations 22

$$\begin{aligned}\boldsymbol{u}_m(\omega) &= -W_\infty\alpha(\omega) - \mathrm{i}\omega\alpha(\omega)(\boldsymbol{z}_c - z_f)\\ \boldsymbol{v}_m(\omega) &= \mathbf{0}\\ \boldsymbol{w}_m(\omega) &= U_\infty\alpha(\omega) + \mathrm{i}\omega\alpha(\omega)(\boldsymbol{x}_c - x_f) + \mathrm{i}\omega h(k)\end{aligned}$$

where θ has been replaced by α for compatibility with Theodorsen theory, q has been replaced by $\mathrm{i}\omega\alpha$ and w has been replaced by $\mathrm{i}\omega h$. All other degrees of freedom have been set to zero. Note that Theodorsen theory ignores the $\boldsymbol{u}_m(\omega)$ velocity and only makes use of the upwash $\boldsymbol{w}_m(\omega)$. Substituting for $\omega = 2kQ_\infty/c_0$ and dividing throughout by Q_∞ leads to

$$\begin{aligned}\bar{\boldsymbol{u}}_m(k) &= -\bar{W}_\infty\alpha(k) - \frac{2\mathrm{i}k}{c_0}\alpha(k)(\boldsymbol{z}_c - z_f)\\ \bar{\boldsymbol{v}}_m(k) &= \mathbf{0}\\ \bar{\boldsymbol{w}}_m(k) &= \bar{U}_\infty\alpha(k) + \frac{2\mathrm{i}k}{c_0}\alpha(k)(\boldsymbol{x}_c - x_f) + \frac{2\mathrm{i}k}{c_0}h(k)\end{aligned} \tag{96}$$

The pressure coefficient at the oscillation frequency, k, is still given by equation 18

$$\begin{aligned}\boldsymbol{c}_p(k) = & -\frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\xi \circ \bar{\boldsymbol{u}}_m(k)) - 2\bar{\boldsymbol{u}}(0)\circ\bar{\boldsymbol{u}}_m(k) - \bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\eta \circ \bar{\boldsymbol{v}}_m(k)) - 2\bar{\boldsymbol{v}}(0)\circ\bar{\boldsymbol{v}}_m(k)\\ & -\bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\zeta \circ \bar{\boldsymbol{w}}_m(k)) - 2\bar{\boldsymbol{w}}(0)\circ\bar{\boldsymbol{w}}_m(k) - \frac{2\mathrm{i}k}{c_0\beta}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\xi \circ \bar{\boldsymbol{u}}_m(k))\\ & -\frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\eta \circ \bar{\boldsymbol{v}}_m(k)) - \frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\zeta \circ \bar{\boldsymbol{w}}_m(k))\end{aligned} \tag{97}$$

Setting $\alpha(k) = 0$ in equations 96 and substituting into equation 97 leads to

$$\boldsymbol{c}_p(k) = -\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\zeta\frac{2\mathrm{i}k}{c_0}h(k)\right) - 2\bar{\boldsymbol{w}}\frac{2\mathrm{i}k}{c_0}h(k) - \frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\zeta\frac{2\mathrm{i}k}{c_0}h(k)\right)$$

Writing the pressure coefficient as

$$\boldsymbol{c}_p(k) = \mathrm{i}k\boldsymbol{c}_{p_{\dot{h}}}(k)h(k) + (\mathrm{i}k)^2\boldsymbol{c}_{p_{\ddot{h}}}(k)h(k)$$

leads to the plunge pressure derivatives

$$\boldsymbol{c}_{p_{\dot{h}}}(k) = -\frac{2}{c_0}\bar{\boldsymbol{C}}_0(k)\boldsymbol{n}_\zeta - \frac{4}{c_0}\bar{\boldsymbol{w}}(0) \tag{98}$$

$$\boldsymbol{c}_{p_{\ddot{h}}}(k) = -\left(\frac{2}{c_0}\right)^2\bar{\boldsymbol{C}}_1(k)\boldsymbol{n}_\zeta \tag{99}$$

Setting $h(k) = 0$ in equations 96 and substituting into equation 97 leads to

$$\begin{aligned}
\boldsymbol{c}_p(k) &= -\frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\xi \circ \left(-\bar{W}_\infty\alpha(k) - \frac{2\mathrm{i}k}{c_0}\alpha(k)(\boldsymbol{z}_c - z_f)\right)\right) \\
&\quad -2\bar{\boldsymbol{u}}(0) \circ \left(-\bar{W}_\infty\alpha(k) - \frac{2\mathrm{i}k}{c_0}\alpha(k)(\boldsymbol{z}_c - z_f)\right) \\
&\quad -\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\zeta \circ \left(\bar{U}_\infty\alpha(k) + \frac{2\mathrm{i}k}{c_0}\alpha(k)(\boldsymbol{x}_c - x_f)\right)\right) \\
&\quad -2\bar{\boldsymbol{w}}(0) \circ \left(\bar{U}_\infty\alpha(k) + \frac{2\mathrm{i}k}{c_0}\alpha(k)(\boldsymbol{x}_c - x_f)\right) \\
&\quad -\frac{2\mathrm{i}k}{c_0\beta}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\xi \circ \left(-\bar{W}_\infty\alpha(k) - \frac{2\mathrm{i}k}{c_0}\alpha(k)(\boldsymbol{z}_c - z_f)\right)\right) \\
&\quad -\frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\zeta \circ \left(\bar{U}_\infty\alpha(k) + \frac{2\mathrm{i}k}{c_0}\alpha(k)(\boldsymbol{x}_c - x_f)\right)\right)
\end{aligned}$$

Finally, writing the pressure coefficient as

$$\boldsymbol{c}_p(k) = \boldsymbol{c}_{p_\alpha}(k)\alpha(k) + \mathrm{i}k\boldsymbol{c}_{p_{\dot\alpha}}(k)\alpha(k) + (\mathrm{i}k)^2\boldsymbol{c}_{p_{\ddot\alpha}}(k)\alpha(k)$$

leads to the plunge pressure derivatives

$$\begin{aligned}
\boldsymbol{c}_{p_\alpha}(k) &= \frac{\bar{W}_\infty}{\beta}\bar{\boldsymbol{C}}_0(k)\boldsymbol{n}_\xi + 2\bar{W}_\infty\bar{\boldsymbol{u}}(0) - \bar{U}_\infty\bar{\boldsymbol{C}}_0(k)\boldsymbol{n}_\zeta - 2\bar{U}_\infty\bar{\boldsymbol{w}}(0) && (100) \\
\boldsymbol{c}_{p_{\dot\alpha}}(k) &= \frac{2}{c_0\beta}\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\xi \circ (\boldsymbol{z}_c - z_f)\right) + \frac{4}{c_0}\bar{\boldsymbol{u}}(0) \circ (\boldsymbol{z}_c - z_f) \\
&\quad -\frac{2}{c_0}\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\zeta \circ (\boldsymbol{x}_c - x_f)\right) - \frac{4}{c_0}\bar{\boldsymbol{w}}(0) \circ (\boldsymbol{x}_c - x_f) \\
&\quad +\frac{2\bar{W}_\infty}{c_0\beta}\bar{\boldsymbol{C}}_1(k)\boldsymbol{n}_\xi - \frac{2\bar{U}_\infty}{c_0}\bar{\boldsymbol{C}}_1(k)\boldsymbol{n}_\zeta && (101) \\
\boldsymbol{c}_{p_{\ddot\alpha}}(k) &= \left(\frac{2}{c_0}\right)^2\frac{1}{\beta}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\xi \circ (\boldsymbol{z}_c - z_f)\right) - \left(\frac{2}{c_0}\right)^2\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\zeta \circ (\boldsymbol{x}_c - x_f)\right) && (102)
\end{aligned}$$

A.3.1 Linearised pressure derivatives for pitching and plunging motion

Equations 98 to 102 give the pressure derivatives obtained from the second order oscillatory pressure equation 97. Their linearized equivalents are obtained from the linearized pressure equation 75

$$\begin{aligned}
\boldsymbol{c}_p(k) &= 2\boldsymbol{K}_x(k)\left(\frac{1}{\beta}\bar{\boldsymbol{u}}_m(k) \circ \boldsymbol{n}_\xi + \bar{\boldsymbol{v}}_m(k) \circ \boldsymbol{n}_\eta + \bar{\boldsymbol{w}}_m(k) \circ \boldsymbol{n}_\zeta\right) \\
&\quad -\frac{4\mathrm{i}k}{c_0}\boldsymbol{K}(k)\left(\frac{1}{\beta}\bar{\boldsymbol{u}}_m(k) \circ \boldsymbol{n}_\xi + \bar{\boldsymbol{v}}_m(k) \circ \boldsymbol{n}_\eta + \bar{\boldsymbol{w}}_m(k) \circ \boldsymbol{n}_\zeta\right) && (103)
\end{aligned}$$

Setting $\alpha(k) = 0$ in equations 96 and substituting into equation 103 leads to

$$\boldsymbol{c}_p(k) = 2\boldsymbol{K}_x(k)\boldsymbol{n}_\zeta\frac{2\mathrm{i}k}{c_0}h(k) - \frac{4\mathrm{i}k}{c_0}\boldsymbol{K}(k)\boldsymbol{n}_\zeta\frac{2\mathrm{i}k}{c_0}h(k)$$

such that the linearized pressure derivatives in the plunge direction become

$$\begin{aligned} \boldsymbol{c}_{p_{\dot h}}(k) &= \frac{4}{c_0}\boldsymbol{K}_x(k)\boldsymbol{n}_\zeta && (104)\\ \boldsymbol{c}_{p_{\ddot h}}(k) &= -\frac{8}{c_0^2}\boldsymbol{K}(k)\boldsymbol{n}_\zeta && (105) \end{aligned}$$

Setting $h(k) = 0$ in equations 96 and substituting into equation 103 results in

$$\begin{aligned} \boldsymbol{c}_p(k) &= \frac{2}{\beta}\boldsymbol{K}_x(k)\left(\left(-\bar{W}_\infty\alpha(k) - \frac{2ik}{c_0}\alpha(k)(\boldsymbol{z}_c - z_f)\right)\circ\boldsymbol{n}_\xi\right)\\ &+2\boldsymbol{K}_x(k)\left(\left(\bar{U}_\infty\alpha(k) + \frac{2ik}{c_0}\alpha(k)(\boldsymbol{x}_c - x_f)\right)\circ\boldsymbol{n}_\zeta\right)\\ &-\frac{4ik}{c_0\beta}\boldsymbol{K}(k)\left(\left(-\bar{W}_\infty\alpha(k) - \frac{2ik}{c_0}\alpha(k)(\boldsymbol{z}_c - z_f)\right)\circ\boldsymbol{n}_\xi\right)\\ &-\frac{4ik}{c_0}\boldsymbol{K}(k)\left(\left(\bar{U}_\infty\alpha(k) + \frac{2ik}{c_0}\alpha(k)(\boldsymbol{x}_c - x_f)\right)\circ\boldsymbol{n}_\zeta\right) \end{aligned}$$

Consequently, the linearized pressure derivatives in the plunge direction are

$$\begin{aligned} \boldsymbol{c}_{p_\alpha}(k) &= -\frac{2\bar{W}_\infty}{\beta}\boldsymbol{K}_x(k)\boldsymbol{n}_\xi + 2\bar{U}_\infty\boldsymbol{K}_x(k)\boldsymbol{n}_\zeta && (106)\\ \boldsymbol{c}_{p_{\dot\alpha}}(k) &= -\frac{4}{c_0\beta}\boldsymbol{K}_x(k)(\boldsymbol{n}_\xi\circ(\boldsymbol{z}_c - z_f)) + \frac{4}{c_0}\boldsymbol{K}_x(k)(\boldsymbol{n}_\zeta\circ(\boldsymbol{x}_c - x_f))\\ &+\frac{4\bar{W}_\infty}{c_0\beta}\boldsymbol{K}(k)\boldsymbol{n}_\xi - \frac{4\bar{U}_\infty}{c_0}\boldsymbol{K}(k)\boldsymbol{n}_\zeta && (107)\\ \boldsymbol{c}_{p_{\ddot\alpha}}(k) &= \frac{8}{c_0^2\beta}\boldsymbol{K}(k)(\boldsymbol{n}_\xi\circ(\boldsymbol{z}_c - z_f)) - \frac{8}{c_0^2}\boldsymbol{K}(k)(\boldsymbol{n}_\zeta\circ(\boldsymbol{x}_c - x_f)) && (108) \end{aligned}$$

A.4 Nonlinear pressure derivatives for flexible motion

The pressure coefficient at the oscillation frequency, k, is still given by equation 18

$$\begin{aligned} \boldsymbol{c}_p(k) &= -\frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\xi\circ\bar{\boldsymbol{u}}_m(k)) - 2\bar{\boldsymbol{u}}(0)\circ\bar{\boldsymbol{u}}_m(k) - \bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\eta\circ\bar{\boldsymbol{v}}_m(k)) - 2\bar{\boldsymbol{v}}(0)\circ\bar{\boldsymbol{v}}_m(k)\\ &-\bar{\boldsymbol{C}}_0(k)(\boldsymbol{n}_\zeta\circ\bar{\boldsymbol{w}}_m(k)) - 2\bar{\boldsymbol{w}}(0)\circ\bar{\boldsymbol{w}}_m(k) - \frac{2ik}{c_0\beta}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\xi\circ\bar{\boldsymbol{u}}_m(k))\\ &-\frac{2ik}{c_0}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\eta\circ\bar{\boldsymbol{v}}_m(k)) - \frac{2ik}{c_0}\bar{\boldsymbol{C}}_1(k)(\boldsymbol{n}_\zeta\circ\bar{\boldsymbol{w}}_m(k)) && (109) \end{aligned}$$

For flexible motion, the motion-induced relative velocities on the surface are given by equations 23

$$\begin{aligned} \bar{\boldsymbol{u}}_m(k) &= \bar{V}_\infty\tilde{\boldsymbol{\Phi}}_\psi\boldsymbol{q}(k) - \bar{W}_\infty\tilde{\boldsymbol{\Phi}}_\theta\boldsymbol{q}(k) - \frac{2ik}{c_0}\tilde{\boldsymbol{\Phi}}_x\boldsymbol{q}(k)\\ \bar{\boldsymbol{v}}_m(k) &= -\bar{U}_\infty\tilde{\boldsymbol{\Phi}}_\psi\boldsymbol{q}(k) + \bar{W}_\infty\tilde{\boldsymbol{\Phi}}_\phi\boldsymbol{q}(k) - \frac{2ik}{c_0}\tilde{\boldsymbol{\Phi}}_y\boldsymbol{q}(k) && (110)\\ \bar{\boldsymbol{w}}_m(k) &= \bar{U}_\infty\tilde{\boldsymbol{\Phi}}_\theta\boldsymbol{q}(k) - \bar{V}_\infty\tilde{\boldsymbol{\Phi}}_\phi\boldsymbol{q}(k) - \frac{2ik}{c_0}\tilde{\boldsymbol{\Phi}}_z\boldsymbol{q}(k) \end{aligned}$$

Setting $\tilde{\boldsymbol{\Phi}}_\phi = \tilde{\boldsymbol{\Phi}}_\theta = \tilde{\boldsymbol{\Phi}}_\psi = \tilde{\boldsymbol{\Phi}}_y = \tilde{\boldsymbol{\Phi}}_z = \mathbf{0}$ and substituting equations 110 into 109 leads to

$$\begin{aligned} \boldsymbol{c}_p(k) &= -\frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\xi \circ \left(-\frac{2\mathrm{i}k}{c_0}\tilde{\boldsymbol{\Phi}}_x \boldsymbol{q}(k)\right)\right) - 2\bar{\boldsymbol{u}}(0) \circ \left(-\frac{2\mathrm{i}k}{c_0}\tilde{\boldsymbol{\Phi}}_x \boldsymbol{q}(k)\right) \\ &\quad -\frac{2\mathrm{i}k}{c_0\beta}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\xi \circ \left(-\frac{2\mathrm{i}k}{c_0}\tilde{\boldsymbol{\Phi}}_x \boldsymbol{q}(k)\right)\right) \end{aligned}$$

We then write $\boldsymbol{c}_p(k)$ in the form

$$\boldsymbol{c}_p(k) = \mathrm{i}k\boldsymbol{c}_{p_{\dot{x}}}(k)\boldsymbol{q}(k) + (\mathrm{i}k)^2\boldsymbol{c}_{p_{\ddot{x}}}(k)\boldsymbol{q}(k)$$

where the pressure derivatives due to motion parallel to the $\tilde{\boldsymbol{\Phi}}_x$ components of the mode shapes are given by

$$\boldsymbol{c}_{p_{\dot{x}}}(k) = \frac{2}{c_0\beta}\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\xi \circ \tilde{\boldsymbol{\Phi}}_x\right) + \frac{4}{c_0}\bar{\boldsymbol{u}}(0) \circ \tilde{\boldsymbol{\Phi}}_x \tag{111}$$

$$\boldsymbol{c}_{p_{\ddot{x}}}(k) = \frac{4}{c_0^2\beta}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\xi \circ \tilde{\boldsymbol{\Phi}}_x\right) \tag{112}$$

Similarly, the pressure derivatives do to motion parallel to the $\tilde{\boldsymbol{\Phi}}_y$ components of the mode shapes become

$$\boldsymbol{c}_{p_{\dot{y}}}(k) = \frac{2}{c_0}\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\eta \circ \tilde{\boldsymbol{\Phi}}_y\right) + \frac{4}{c_0}\bar{\boldsymbol{v}}(0) \circ \tilde{\boldsymbol{\Phi}}_y \tag{113}$$

$$\boldsymbol{c}_{p_{\ddot{y}}}(k) = \frac{4}{c_0^2}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\eta \circ \tilde{\boldsymbol{\Phi}}_y\right) \tag{114}$$

while the pressure derivatives in the $\tilde{\boldsymbol{\Phi}}_z$ direction are

$$\boldsymbol{c}_{p_{\dot{z}}}(k) = \frac{2}{c_0}\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\zeta \circ \tilde{\boldsymbol{\Phi}}_z\right) + \frac{4}{c_0}\bar{\boldsymbol{w}}(0) \circ \tilde{\boldsymbol{\Phi}}_z \tag{115}$$

$$\boldsymbol{c}_{p_{\ddot{z}}}(k) = \frac{4}{c_0^2}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\zeta \circ \tilde{\boldsymbol{\Phi}}_z\right) \tag{116}$$

Next, we set $\tilde{\boldsymbol{\Phi}}_\theta = \tilde{\boldsymbol{\Phi}}_\psi = \tilde{\boldsymbol{\Phi}}_x = \tilde{\boldsymbol{\Phi}}_y = \tilde{\boldsymbol{\Phi}}_z = \mathbf{0}$ and substitute equations 110 into 109 to obtain

$$\begin{aligned} \boldsymbol{c}_p(k) &= -\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\eta \circ \left(\bar{W}_\infty \tilde{\boldsymbol{\Phi}}_\phi \boldsymbol{q}(k)\right)\right) - 2\bar{\boldsymbol{v}}(0) \circ \left(\bar{W}_\infty \tilde{\boldsymbol{\Phi}}_\phi \boldsymbol{q}(k)\right) \\ &\quad -\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\zeta \circ \left(-\bar{V}_\infty \tilde{\boldsymbol{\Phi}}_\phi \boldsymbol{q}(k)\right)\right) - 2\bar{\boldsymbol{w}}(0) \circ \left(-\bar{V}_\infty \tilde{\boldsymbol{\Phi}}_\phi \boldsymbol{q}(k)\right) \\ &\quad -\frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\eta \circ \left(\bar{W}_\infty \tilde{\boldsymbol{\Phi}}_\phi \boldsymbol{q}(k)\right)\right) - \frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\zeta \circ \left(-\bar{V}_\infty \tilde{\boldsymbol{\Phi}}_\phi \boldsymbol{q}(k)\right)\right) \end{aligned}$$

The pressure coefficient equation is written as

$$\boldsymbol{c}_p(k) = \boldsymbol{c}_{p_\phi}(k)\boldsymbol{q}(k) + \mathrm{i}k\boldsymbol{c}_{p_{\dot{\phi}}}(k)\boldsymbol{q}(k)$$

where

$$\begin{aligned}
\boldsymbol{c}_{p_\phi}(k) &= -\bar{W}_\infty \bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\eta \circ \tilde{\boldsymbol{\Phi}}_\phi\right) - 2\bar{W}_\infty \bar{\boldsymbol{v}}(0) \circ \tilde{\boldsymbol{\Phi}}_\phi \\
&\quad + \bar{V}_\infty \bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\zeta \circ \tilde{\boldsymbol{\Phi}}_\phi\right) + 2\bar{V}_\infty \bar{\boldsymbol{w}}(0) \circ \tilde{\boldsymbol{\Phi}}_\phi \qquad (117) \\
\boldsymbol{c}_{p_{\dot\phi}}(k) &= -\frac{2\bar{W}_\infty}{c_0}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\eta \circ \tilde{\boldsymbol{\Phi}}_\phi\right) + \frac{2\bar{V}_\infty}{c_0}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\zeta \circ \tilde{\boldsymbol{\Phi}}_\phi\right) \qquad (118)
\end{aligned}$$

Again, we set $\tilde{\boldsymbol{\Phi}}_\phi = \tilde{\boldsymbol{\Phi}}_\psi = \tilde{\boldsymbol{\Phi}}_x = \tilde{\boldsymbol{\Phi}}_y = \tilde{\boldsymbol{\Phi}}_z = \boldsymbol{0}$ and substitute equations 110 into 109 to obtain

$$\begin{aligned}
\boldsymbol{c}_p(k) &= -\frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\xi \circ \left(-\bar{W}_\infty \tilde{\boldsymbol{\Phi}}_\theta \boldsymbol{q}(k)\right)\right) - 2\bar{\boldsymbol{u}}(0) \circ \left(-\bar{W}_\infty \tilde{\boldsymbol{\Phi}}_\theta \boldsymbol{q}(k)\right) \\
&\quad - \bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\zeta \circ \left(\bar{U}_\infty \tilde{\boldsymbol{\Phi}}_\theta \boldsymbol{q}(k)\right)\right) - 2\bar{\boldsymbol{w}}(0) \circ \left(\bar{U}_\infty \tilde{\boldsymbol{\Phi}}_\theta \boldsymbol{q}(k)\right) \\
&\quad - \frac{2\mathrm{i}k}{c_0\beta}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\xi \circ \left(-\bar{W}_\infty \tilde{\boldsymbol{\Phi}}_\theta \boldsymbol{q}(k)\right)\right) - \frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\zeta \circ \left(\bar{U}_\infty \tilde{\boldsymbol{\Phi}}_\theta \boldsymbol{q}(k)\right)\right)
\end{aligned}$$

so that the pressure derivatives in the $\tilde{\boldsymbol{\Phi}}_\theta$ direction become

$$\begin{aligned}
\boldsymbol{c}_{p_\theta}(k) &= \frac{\bar{W}_\infty}{\beta}\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\xi \circ \tilde{\boldsymbol{\Phi}}_\theta\right) + 2\bar{W}_\infty \bar{\boldsymbol{u}}(0) \circ \tilde{\boldsymbol{\Phi}}_\theta \\
&\quad - \bar{U}_\infty \bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\zeta \circ \tilde{\boldsymbol{\Phi}}_\theta\right) - 2\bar{U}_\infty \bar{\boldsymbol{w}}(0) \circ \tilde{\boldsymbol{\Phi}}_\theta \qquad (119) \\
\boldsymbol{c}_{p_{\dot\theta}}(k) &= \frac{2\bar{W}_\infty}{c_0\beta}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\xi \circ \tilde{\boldsymbol{\Phi}}_\theta\right) - \frac{2\bar{U}_\infty}{c_0}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\zeta \circ \tilde{\boldsymbol{\Phi}}_\theta\right) \qquad (120)
\end{aligned}$$

Finally, setting $\tilde{\boldsymbol{\Phi}}_\phi = \tilde{\boldsymbol{\Phi}}_\theta = \tilde{\boldsymbol{\Phi}}_x = \tilde{\boldsymbol{\Phi}}_y = \tilde{\boldsymbol{\Phi}}_z = \boldsymbol{0}$ and repeating the procedure yields

$$\begin{aligned}
\boldsymbol{c}_p(k) &= -\frac{1}{\beta}\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\xi \circ \left(\bar{V}_\infty \tilde{\boldsymbol{\Phi}}_\psi \boldsymbol{q}(k)\right)\right) - 2\bar{\boldsymbol{u}}(0) \circ \left(\bar{V}_\infty \tilde{\boldsymbol{\Phi}}_\psi \boldsymbol{q}(k)\right) \\
&\quad - \bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\eta \circ \left(-\bar{U}_\infty \tilde{\boldsymbol{\Phi}}_\psi \boldsymbol{q}(k)\right)\right) - 2\bar{\boldsymbol{v}}(0) \circ \left(-\bar{U}_\infty \tilde{\boldsymbol{\Phi}}_\psi \boldsymbol{q}(k)\right) \\
&\quad - \frac{2\mathrm{i}k}{c_0\beta}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\xi \circ \left(\bar{V}_\infty \tilde{\boldsymbol{\Phi}}_\psi \boldsymbol{q}(k)\right)\right) - \frac{2\mathrm{i}k}{c_0}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\eta \circ \left(-\bar{U}_\infty \tilde{\boldsymbol{\Phi}}_\psi \boldsymbol{q}(k)\right)\right)
\end{aligned}$$

so that the pressure derivatives in the $\tilde{\boldsymbol{\Phi}}_\psi$ direction become

$$\begin{aligned}
\boldsymbol{c}_{p_\psi}(k) &= -\frac{\bar{V}_\infty}{\beta}\bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\xi \circ \tilde{\boldsymbol{\Phi}}_\psi \boldsymbol{q}(k)\right) - 2\bar{V}_\infty \bar{\boldsymbol{u}}(0) \circ \tilde{\boldsymbol{\Phi}}_\psi \\
&\quad + \bar{U}_\infty \bar{\boldsymbol{C}}_0(k)\left(\boldsymbol{n}_\eta \circ \tilde{\boldsymbol{\Phi}}_\psi\right) + 2\bar{U}_\infty \bar{\boldsymbol{v}}(0) \circ \tilde{\boldsymbol{\Phi}}_\psi \qquad (121) \\
\boldsymbol{c}_{p_{\dot\psi}}(k) &= -\frac{2\bar{V}_\infty}{c_0\beta}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\xi \circ \tilde{\boldsymbol{\Phi}}_\psi\right) + \frac{2\bar{U}_\infty}{c_0}\bar{\boldsymbol{C}}_1(k)\left(\boldsymbol{n}_\eta \circ \tilde{\boldsymbol{\Phi}}_\psi\right) \qquad (122)
\end{aligned}$$

B Doublet Lattice Method

B.1 Evaluation of the Kernel function

Equation 35 can be written in the form

$$K(x_0, y_0, z_0, \omega) = \mathrm{e}^{-\mathrm{i}\omega x_0/U_\infty} \frac{\partial^2}{\partial z^2} I_0 \tag{123}$$

where

$$I_0 = \int_{-\infty}^{x_0} \frac{\mathrm{e}^{\frac{\mathrm{i}\omega}{U_\infty \beta^2}(\lambda - M_\infty r_\beta(\lambda, y_0, z_0))}}{r_\beta(\lambda, y_0, z_0)} \mathrm{d}\lambda \tag{124}$$

The integral I_0 is evaluated by means of the variable substitution

$$v = \frac{\lambda}{\beta r} \tag{125}$$

where $r = \sqrt{y_0^2 + z_0^2}$ and $\beta r = r_\beta(0, y_0, z_0)$. Substituting into equation 124 leads to

$$I_0 = \int_{-\infty}^{x_0/\beta r} \frac{1}{\sqrt{1+v^2}} \mathrm{e}^{\frac{\mathrm{i}k_1}{\beta}\left(v - M_\infty \sqrt{1+v^2}\right)} \mathrm{d}v \tag{126}$$

where k_1 is the reduced frequency based on r

$$k_1 = \frac{\omega r}{U_\infty} \tag{127}$$

The integral in expression 126 cannot be evaluated analytically, so that a second variable substituting, that is

$$u = -\frac{1}{\beta}\left(v - M_\infty \sqrt{1+v^2}\right) \tag{128}$$

Substituting into equation 126 results in

$$I_0 = \int_{u_1}^{\infty} \frac{\mathrm{e}^{-\mathrm{i}k_1 u}}{\sqrt{1+u^2}} \mathrm{d}u \tag{129}$$

where

$$u_1(x_0, y_0, z_0) = \frac{-x_0 + M_\infty r_\beta(x_0, y_0, z_0)}{\beta^2 r} \tag{130}$$

and

$$r_\beta(x_0, y_0, z_0) = \sqrt{x_0^2 + \beta^2 y_0^2 + \beta^2 z_0^2} = \sqrt{x_0^2 + \beta^2 r^2}$$

Following [10], the differentiation in equation 123 is carried out such that

$$K(x_0, y_0, z_0, \omega) = \mathrm{e}^{-\mathrm{i}\omega x_0/U_\infty} \frac{K_1 T_1 + K_2 T_2}{r^2} \tag{131}$$

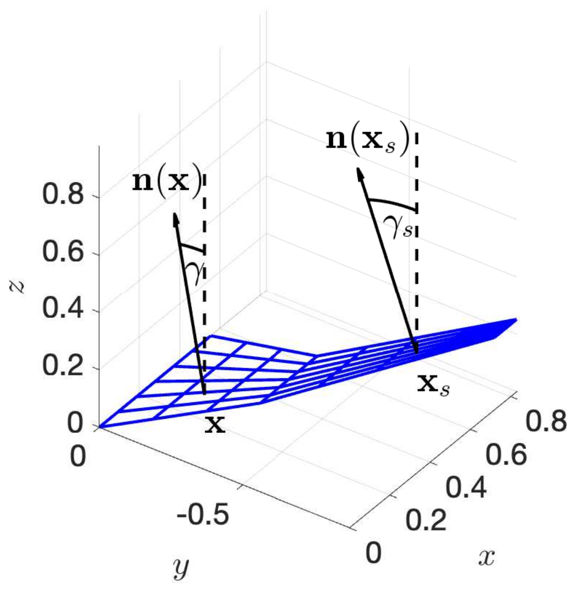


Figure 15: Definition of angles γ and γ_s

where T_1 and T_2 are functions of the angle between the normal vector to the surface and the vertical at the influenced point γ and at the influencing point γ_s. The expressions for T_1 and T_2 are

$$T_1 = \cos(\gamma - \gamma_s) \tag{132}$$

$$T_2 = \left(\frac{z_0}{r}\cos\gamma - \frac{y_0}{r}\sin\gamma\right)\left(\frac{z_0}{r}\cos\gamma_s - \frac{y_0}{r}\sin\gamma_s\right) \tag{133}$$

Figure 15 plots a wing composed of two trapezoidal sections with non-zero dihedral, idealised in the usual DLM way: the wing is infinitely thin and parallel to the x axis in the chordwise direction. Camber, thickness and twist are not represented geometrically. The unit normal vector to the surface, $\mathbf{n} = (n_x,\ n_y,\ n_z)$, is plotted at two points, the influenced point $\mathbf{x}$ and the influencing point $\mathbf{x}_s$. As the wing is flat in the x direction, the x component of all normal vectors is equal to zero and the angles γ and γ_s are defined in the $y - z$ plane. These angles are calculated from the vector products

$$\cos\gamma = -\operatorname{sgn}(n_y(\mathbf{x}))\mathbf{n}(\mathbf{x}) \cdot (0,\ 0,\ 1)\,,\ \cos\gamma_s = -\operatorname{sgn}(n_y(\mathbf{x}))\mathbf{n}(\mathbf{x}_s) \cdot (0,\ 0,\ 1)$$

where sgn is the signum function and is necessary for two-sided wing cases. The negative sign is used because, in the present case, the x axis is defined positive in the flow direction, the z axis positive upwards and the y axis positive using the right-hand rule. In the classic DLM derivation, the x axis is defined positive in a direction opposite to the flow, the z axis positive upwards and the y axis positive using the right-hand rule. Consequently, the positive directions of the x and y axes are opposite in the two implementations.

The term K_1 in equation 131 is given by

$$K_1 = I_1 + \frac{M_\infty r}{r_\beta}\frac{\mathrm{e}^{-\mathrm{i}k_1u_1}}{\sqrt{1+u_1^2}} \tag{134}$$

where

$$I_1 = \int_{u_1}^{\infty} \frac{\mathrm{e}^{-\mathrm{i}k_1 u}}{(1+u^2)^{3/2}} \mathrm{d}u \tag{135}$$

Finally, the term K_2 is given by

$$\begin{aligned} K_2 &= -3I_2 - \frac{\mathrm{i}k_1 M_\infty^2 r^2}{r_\beta^2} \frac{\mathrm{e}^{-\mathrm{i}k_1 u_1}}{\sqrt{1+u_1^2}} \\ &\quad - \frac{M_\infty r}{r_\beta} \left((1+u_1^2)\frac{\beta^2 r^2}{r_\beta^2} + 2 + \frac{M_\infty r u_1}{r_\beta} \right) \frac{\mathrm{e}^{-\mathrm{i}k_1 u_1}}{(1+u_1^2)^{3/2}} \end{aligned} \tag{136}$$

where

$$I_2 = \int_{u_1}^{\infty} \frac{\mathrm{e}^{-\mathrm{i}k_1 u}}{(1+u^2)^{5/2}} \mathrm{d}u \tag{137}$$

It should be stated that [7] gives opposite signs for K_1 and K_2.

The fundamental problem for acceleration potential approaches lies in calculating the I_1 and I_2 integrals, assembling the Kernel function using equation 131 and then substituting it in equation 34 to calculate the pressure jump over the surface as a function of the upwash. In the planar case, $\gamma = \gamma_s = 0$, so that $T_1 = 1$ and $T_2 = (z_0/r)^2$. However, as the upwash equation is written for point $\mathbf{x}$ lying on the surface, $z_0 = 0$ and $T_2 = 0$. Consequently, the Kernel function of expression 131 is simplified to

$$K(x_0, y_0, \omega) = \mathrm{e}^{-\mathrm{i}\omega x_0/U_\infty} \frac{K_1 T_1}{r^2} \tag{138}$$

and only the integral I_1 is required.

B.1.1 Approximation of the Kernel function

[7] shows that, after assuming that $u_1 \geq 0$ and using integration by parts, equation 135 can be written as

$$I_1 = \mathrm{e}^{-\mathrm{i}k_1 u_1} \left(1 - \frac{u_1}{\sqrt{1+u_1^2}} - \mathrm{i}k_1 J_1 \right) \tag{139}$$

where

$$J_1 = \mathrm{e}^{\mathrm{i}k_1 u_1} \int_{u_1}^{\infty} \left(1 - \frac{u}{\sqrt{1+u^2}} \right) \mathrm{e}^{-\mathrm{i}k_1 u} \mathrm{d}u \tag{140}$$

The J_1 integrals cannot be evaluated analytically but is usually approximated by means of an exponential series. The most common such series is attributed to Laschka but the obscurity of the original references means that most people quote appendix A of [8]. [32] generalised the Laschka series and developped higher order approximations. The Laschka approximation states that

$$1 - \frac{u}{\sqrt{1+u^2}} \approx \sum_{n=1}^{11} a_n \mathrm{e}^{-ncu} \tag{141}$$

where $c = 0.372$ and a_n are the elements of the vector

$$\mathbf{a} = \begin{pmatrix} 0.24186198 \\ -2.7918027 \\ 24.991079 \\ -111.59196 \\ 271.43549 \\ -305.75288 \\ -41.183630 \\ 545.98537 \\ -644.78155 \\ 328.72755 \\ -64.279511 \end{pmatrix}$$

Substituting this approximation in equation 140 results in an integral that can be evaluated analytically such that

$$J_1 \approx \sum_{n=1}^{11} \frac{a_n \mathrm{e}^{-ncu_1}}{n^2c^2 + k_1^2}(nc - \mathrm{i}k_1) \tag{142}$$

Finally, substituting equation 142 into expression 139 yields the value of the I_1 integral.

For the I_2 integral of equation 137, [8] state that, for $u_1 \geq 0$ and after applying integration by parts twice, it becomes

$$I_2 = \frac{1}{3}\mathrm{e}^{-\mathrm{i}k_1u_1}\left((2 + \mathrm{i}k_1u_1)\left(1 - \frac{u_1}{\sqrt{1+u_1^2}}\right) - \frac{u_1}{(1+u^2)^{3/2}} - \mathrm{i}k_1J_1 + k_1^2J_2\right) \tag{143}$$

where

$$J_2 = \mathrm{e}^{\mathrm{i}k_1u_1}\int_{u_1}^{\infty} u\left(1 - \frac{u}{\sqrt{1+u^2}}\right)\mathrm{e}^{-\mathrm{i}k_1u}\mathrm{d}u \tag{144}$$

Applying the approximation of equation 141 leads to

$$J_2 \approx \sum_{n=1}^{11} \frac{a_n \mathrm{e}^{-ncu_1}}{(n^2c^2 + k_1^2)^2}\left[n^2c^2 - k_1^2 + ncu_1(n^2c^2 + k_1^2) - \mathrm{i}k_1\left(2nc + u_1(n^2c^2 + k_1^2)\right)\right] \tag{145}$$

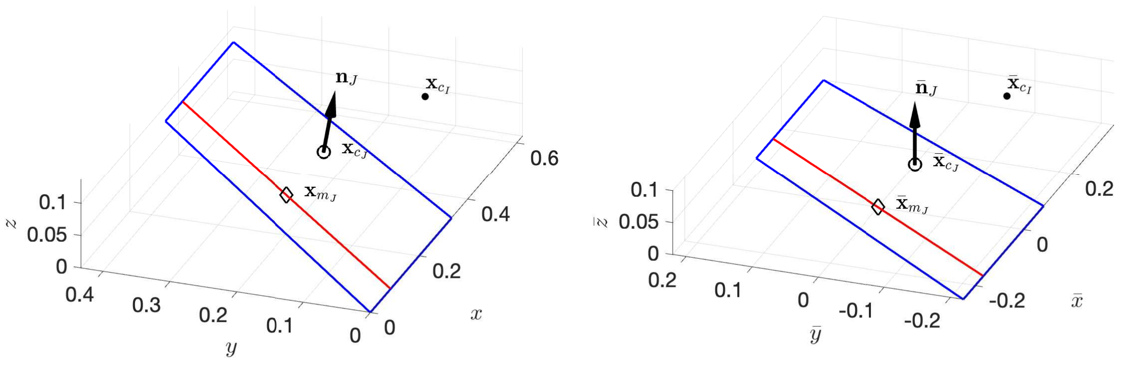
Substituting equations 142 and 145 into expression 143 results in the value of the I_2 integral. It should be recalled that both equations 139 and 143 were obtained for $u_1 \geq 0$. In the case $u_1 < 0$, [8] state that the symmetry properties of the integrands can be used to show that

$$\begin{aligned} I_1(u_1, k_1) &= 2\Re(I_1(0, k_1)) - \Re(I_1(-u_1, k_1)) + \mathrm{i}\Im(I_1(-u_1, k_1)) & (146) \\ I_2(u_1, k_1) &= 2\Re(I_2(0, k_1)) - \Re(I_2(-u_1, k_1)) + \mathrm{i}\Im(I_2(-u_1, k_1)) & (147) \end{aligned}$$

so that both I_1 and I_2 can be obtained from their values at $u_1 \geq 0$.

B.2 Evaluating the planar upwash factor

Even after substituting for the values of K_1 and T_1 in equation 39 there is no known analytical evaluation of the integral. The approach adopted for the DLM is to replace the numerator of the integral by a polynomial function of the spanwise coordinate. [10] states that the initial polynomial approximation was quadratic [33, 8] but a quartic approximation was adopted later [13].



(a) Global coordinates (b) Panel coordinates

Figure 16: Coordinate transformation from global to panel coordinates

First, the coordinates of the influencing panel, J, and influenced point, I, are rotated to the frame of reference of the influencing panel, centred at the midpoint of its doublet line, $\mathbf{x}_{m_J}$, as shown in figure 16. The panel coordinate system is defined as

$$\begin{aligned}
\bar{x} &= x - x_{m_J} \\
\bar{y} &= (y - y_{m_J})\cos\gamma_J + (z - z_{m_J})\sin\gamma_J \\
\bar{z} &= (z - z_{m_J})\cos\gamma_J - (y - y_{m_J})\sin\gamma_J \\
\bar{\gamma} &= \gamma - \gamma_J
\end{aligned}$$

Figure 16(a) plots the Jth panel and Ith control point in the global coordinates. The midpoint of the doublet line is denoted by $\mathbf{x}_{m_J}$, the control point by $\mathbf{x}_{c_J}$ and the normal unit vector by $\mathbf{n}_J$, whose angle to the vertical is denoted by γ_J. The influence of this panel is sought at the Ith control point $\mathbf{x}_{c_I}$. Figure 16(b) plots the Jth panel and Ith control point in the panel coordinates. The midpoint of the doublet line, $\bar{\mathbf{x}}_{m_J}$ now lies on the origin, the control point $\bar{\mathbf{x}}_{c_J}$ lies on $\bar{y} = 0$, the transformed normal vector is $\bar{\mathbf{n}}_J = (0,\ 0,\ 1)$ so that $\bar{\gamma}_J = 0$ and the entire panel lies on the $\bar{z} = 0$ plane. The influenced point now lies on $\bar{\mathbf{x}}_{c_I}$ and the local dihedral angle is $\bar{\gamma}_I = \gamma_I - \gamma_J$.

The geometry of the panel in its own coordinate system is further explained in figure 17. In particular, the sweep angle Λ_J is defined from the $-\bar{y}$ axis and its positivity is determined using the right-hand rule along the positive $\bar{z}$ axis. The mean chordwise length of the panel is $\Delta\bar{x}_J$ and its mean spanwise length $\Delta\bar{y}_J$, so that the $\bar{y}$ coordinate

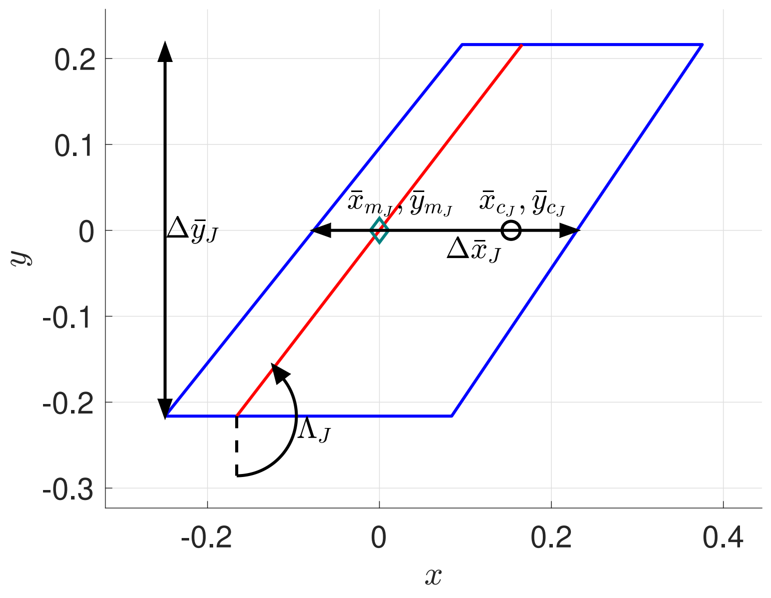


Figure 17: 2D view of Jth panel in its own coordinate system

ranges from $-\Delta\bar{y}_J/2$ to $\Delta\bar{y}_J/2$. The panel aspect ratio is defined as

$$\text{Æ}_J = \frac{\Delta \bar{y}_J}{\Delta \bar{x}_J} \tag{148}$$

On the doublet line, the $\bar{x}$ coordinate is related to $\bar{y}$ by the sweep angle, such that

$$\bar{x} = \bar{y} \tan \Lambda_J$$

Looking at the Kernel function of equation 123, the lengths x_0, y_0, z_0 concerning the Ith control point and the Jth panel become

$$x_0(\bar{y}) = \bar{x}_I - \bar{x} = \bar{x}_I - \bar{y} \tan \Lambda_J, \;\; y_0(\bar{y}) = \bar{y}_I - \bar{y}, \;\; z_0 = \bar{z}_I$$

where $\bar{x}$, $\bar{y}, 0$ are the coordinates of the doublet line of the Jth panel. The distance r in equation 125 becomes

$$r(\bar{y}) = \sqrt{(\bar{y}_I - \bar{y})^2 + \bar{z}_I^2}$$

the reduced frequency k_1 defined in expression 127 is

$$k_1(\bar{y}, \omega) = \frac{\omega r(\bar{y})}{U_\infty}$$

while the elliptical distance r_β in equation 34 becomes

$$r_\beta(\bar{y}) = \sqrt{x_0(\bar{y})^2 + \beta^2 r(\bar{y})^2}$$

and the lower limit u_1 in equation 129 is

$$u_1(\bar{y}) = \frac{-x_0(\bar{y}) + M_\infty r_\beta(\bar{y})}{r(\bar{y})\beta^2}$$

Note that, when $I = J$, i.e. when the influenced point is the control point of the influenced panel, $r(0) = 0$ at the midpoint of the doublet line and $u_1(0)$ tends to infinity. Generalising the workaround proposed by [7] to nonplanar cases, $r(0)$ is set to $r(0) = 10^{-12}$ when $|r(0)| < 10^{-12}$.

Finally, T_1 in equation 132 is a constant,

$$T_1 = \cos\bar{\gamma}_I$$

It follows that J_1 in equation 142, I_1 in equation 139 and K_1 in equation 134 are also functions of $\bar{y}$ and ω only, so that equation 39 becomes

$$D_{1_{I,J}}(\omega) = \Delta\bar{x}_J \int_{-\Delta\bar{y}_J/2}^{\Delta\bar{y}_J/2} \mathrm{e}^{-\mathrm{i}\omega x_0(\bar{y})/U_\infty} \frac{K_1(\bar{y},\omega)T_1}{r(\bar{y})^2} \mathrm{d}\bar{y}$$

The numerator of the integrand in this latest expression is approximated by the quadratic polynomial [8]

$$D_{1_{I,J}}(\omega) = \Delta\bar{x}_J \int_{-\Delta\bar{y}_J/2}^{\Delta\bar{y}_J/2} \frac{A_1\bar{y}^2 + B_1\bar{y} + C_1}{r(\bar{y})^2} \mathrm{d}\bar{y} \tag{149}$$

where

$$P_1(\bar{y},\omega) = \mathrm{e}^{-\mathrm{i}\omega x_0(\bar{y})/U_\infty} K_1(\bar{y},\omega)T_1 \approx A_1\bar{y}^2 + B_1\bar{y} + C_1 \tag{150}$$

and the A_1, B_1, C_1 complex coefficients are given by

$$\begin{aligned}
A_1 &= \frac{P_1(-\Delta\bar{y}_J/2) - 2P_1(0) + P_1(\Delta\bar{y}_J/2)}{2(\Delta\bar{y}_J/2)^2} \\
B_1 &= \frac{P_1(\Delta\bar{y}_J/2) - P_1(-\Delta\bar{y}_J/2)}{\Delta\bar{y}_J} \\
C_1 &= P_1(0)
\end{aligned}$$

Clearly, the coefficients of the polynomial are chosen such that its values are exact at the two ends and the midpoint of the doublet line. For this quadratic polynomial approximation to be accurate, the panel aspect ratio $\mathit{Æ}_J$ (see equation 148) should be less than 3 [13] or 4 [7, 9]. Equation 149 contains three indefinite integrals that can be evaluated analytically, that is

$$\begin{aligned}
\int \frac{1}{(\bar{y}_I - \bar{y})^2 + \bar{z}_I^2} \mathrm{d}\bar{y} &= \frac{1}{\bar{z}_I} \tan^{-1}\left(\frac{\bar{y} - \bar{y}_I}{\bar{z}_I}\right) \\
\int \frac{\bar{y}}{(\bar{y}_I - \bar{y})^2 + \bar{z}_I^2} \mathrm{d}\bar{y} &= \frac{1}{2} \ln\left((\bar{y}_I - \bar{y})^2 + \bar{z}_I^2\right) + \frac{\bar{y}_I}{\bar{z}_I} \tan^{-1}\left(\frac{\bar{y} - \bar{y}_I}{\bar{z}_I}\right) \\
\int \frac{\bar{y}}{(\bar{y}_I - \bar{y})^2 + \bar{z}_I^2} \mathrm{d}\bar{y} &= \bar{y} + \bar{y}_I \ln\left((\bar{y}_I - \bar{y})^2 + \bar{z}_I^2\right) - \left(\bar{z}_I - \frac{\bar{y}^2}{\bar{z}_I}\right) \tan^{-1}\left(\frac{\bar{y} - \bar{y}_I}{\bar{z}_I}\right)
\end{aligned}$$

The limits of integration are applied using the trigonometric identity

$$\tan^{-1} a - \tan^{-1} b = \tan^{-1} \frac{a-b}{1+ab}$$

so that equation 149 finally becomes

$$D_{1_{I,J}}(\omega) = \Delta\bar{x}_J \left[\left((\bar{y}_I^2 - \bar{z}_I^2)A_1 + \bar{y}_I B_1 + C_1 \right) F + \left(\frac{B_1}{2} + \bar{y}_I A_1 \right) G + \Delta\bar{y}_J A_1 \right] \tag{151}$$

where

$$F = \frac{1}{|\bar{z}_I|} \tan^{-1} \left(\frac{\Delta\bar{y}_J |\bar{z}_I|}{\bar{y}_I^2 + \bar{z}_I^2 - \Delta\bar{y}_J^2/4} \right) \tag{152}$$

$$G = \ln \left(\frac{(\bar{y}_I - \Delta\bar{y}_J/2)^2 + \bar{z}_I^2}{(\bar{y}_I + \Delta\bar{y}_J/2)^2 + \bar{z}_I^2} \right) \tag{153}$$

The planar upwash factor has now been evaluated but there is a singularity; when $\bar{z}_I$ approaches zero, F becomes indeterminate. In order to overcome the singularity, [8] use the arctangent series

$$\tan^{-1} \varepsilon = \sum_{n=0}^{\infty} \frac{(-1)^n \varepsilon^{2n+1}}{2n+1}$$

where

$$\varepsilon = \frac{\Delta\bar{y}_J |\bar{z}_I|}{\bar{y}_I^2 + \bar{z}_I^2 - \Delta\bar{y}_J^2/4} \tag{154}$$

Substituting this series up to eleventh order in equation 152 leads to

$$F \approx \frac{\Delta\bar{y}_J}{\bar{y}_I^2 + \bar{z}_I^2 - \Delta\bar{y}_J^2/4} \left(1 - a \frac{\bar{z}_I^2}{\Delta\bar{y}_J^2/4} \right) \tag{155}$$

where

$$a = \frac{\Delta\bar{y}_J^4/4}{(\bar{y}_I^2 + \bar{z}_I^2 - \Delta\bar{y}_J^2/4)^2} \sum_{n=2}^{7} \frac{(-1)^n}{2n-1} \varepsilon^{2n-4} \tag{156}$$

Equation 152 is used for cases where

$$|\varepsilon| > 0.3 \text{ and } \frac{|\bar{z}_I|}{\Delta\bar{y}/2} > 0.001$$

while equation 155 is employed when

$$|\varepsilon| \leq 0.3 \text{ and } \frac{|\bar{z}_I|}{\Delta\bar{y}/2} > 0.001$$

Cases for which

$$\frac{|\bar{z}_I|}{\Delta\bar{y}/2} \leq 0.001$$

are assumed to planar[3], i.e. $\bar{z}_I = 0$. Taking the limit of equation 152 as $\bar{z}_I \to 0$ or setting $a = 0$ in equation 155 leads to

$$F = \frac{\Delta\bar{y}_J}{\bar{y}_I^2 - \Delta\bar{y}_J^2/4} \tag{157}$$

[3]This information was inferred from the Nastran source code by [9]

B.3 Evaluating the nonplanar upwash factor

The procedure used for the evaluation of term $D_{2_{I,J}}(\omega)$ in equation 38 is similar to that used for term $D_{1_{I,J}}(\omega)$. Recall that

$$D_{2_{I,J}}(\omega) = \Delta\bar{x}_J \int_{S_J} \mathrm{e}^{-\mathrm{i}\omega x_0/U_\infty} \frac{K_2 T_2^*}{r^4} \mathrm{d}y_s \tag{158}$$

and that the term K_2 is given by equation 136, while

$$T_2^* = (z_0 \cos\gamma - y_0 \sin\gamma)(z_0 \cos\gamma_s - y_0 \sin\gamma_s) \tag{159}$$

Both K_2 and T_2^* are functions of $\bar{y}$ and ω only, so that equation 158 becomes

$$D_{2_{I,J}}(\omega) = \Delta\bar{x}_J \int_{-\Delta\bar{y}_J/2}^{\Delta\bar{y}_J/2} \mathrm{e}^{-\mathrm{i}\omega x_0(\bar{y})/U_\infty} \frac{K_2(\bar{y},\omega) T_2^*(\bar{y})}{r(\bar{y})^4} \mathrm{d}\bar{y}$$

Again, the numerator in the integrand above is approximated by a second order polynomial, such that

$$D_{12_{I,J}}(\omega) = \Delta\bar{x}_J \int_{-\Delta\bar{y}_J/2}^{\Delta\bar{y}_J/2} \frac{A_2\bar{y}^2 + B_2\bar{y} + C_2}{r(\bar{y})^4} \mathrm{d}\bar{y} \tag{160}$$

where

$$P_2(\bar{y}) = \mathrm{e}^{-\mathrm{i}\omega x_0(\bar{y})/U_\infty} K_1(\bar{y},\omega) T_2^* \approx A_2\bar{y}^2 + B_2\bar{y} + C_2 \tag{161}$$

and the A_2, B_2, C_2 complex coefficients are given by

$$\begin{aligned}
A_2 &= \frac{P_2(-\Delta\bar{y}_J/2) - 2P_2(0) + P_2(\Delta\bar{y}_J/2)}{2(\Delta\bar{y}_J/2)^2} \\
B_2 &= \frac{P_2(\Delta\bar{y}_J/2) - P_2(-\Delta\bar{y}_J/2)}{\Delta\bar{y}_J} \\
C_2 &= P_2(0)
\end{aligned}$$

The three integrals in equation 160 have the analytical indefinite forms

$$\begin{aligned}
\int \frac{1}{\left((\bar{y}_I - \bar{y})^2 + \bar{z}_I^2\right)^2} \mathrm{d}\bar{y} &= \frac{1}{2\bar{z}_I^3} \tan^{-1}\left(\frac{\bar{y} - \bar{y}_I}{\bar{z}_I}\right) + \frac{\bar{y} - \bar{y}_I}{2\bar{z}_I^2\left((\bar{y} - \bar{y}_I)^2 + \bar{z}_I^2\right)} \\
\int \frac{\bar{y}}{\left((\bar{y}_I - \bar{y})^2 + \bar{z}_I^2\right)^2} \mathrm{d}\bar{y} &= \frac{\bar{y}_I}{2\bar{z}_I^3} \tan^{-1}\left(\frac{\bar{y} - \bar{y}_I}{\bar{z}_I}\right) - \frac{\bar{y}_I^2 - \bar{y}\bar{y}_I + \bar{z}_I^2}{2\bar{z}_I^2\left((\bar{y} - \bar{y}_I)^2 + \bar{z}_I^2\right)} \\
\int \frac{\bar{y}^2}{\left((\bar{y}_I - \bar{y})^2 + \bar{z}_I^2\right)^2} \mathrm{d}\bar{y} &= \frac{\bar{y}_I^2 + \bar{z}_I^2}{2\bar{z}_I^3} \tan^{-1}\left(\frac{\bar{y} - \bar{y}_I}{\bar{z}_I}\right) - \frac{\bar{y}_I^3 - \bar{y}\bar{y}_I^2 + (\bar{y} + \bar{y}_I)\bar{z}_I^2}{2\bar{z}_I^2\left((\bar{y} - \bar{y}_I)^2 + \bar{z}_I^2\right)}
\end{aligned}$$

After application of the limits of integration, equation 160 becomes

$$D_{2_{I,J}}(\omega) = \frac{\Delta\bar{x}_J}{2\bar{z}_I^2}(KA_2 + LB_2 + MC_2) \tag{162}$$

where

$$
\begin{aligned}
K &= \left(\bar{y}_I^2+\bar{z}_I^2\right)F + \frac{(\bar{y}_I^2+\bar{z}_I^2)\bar{y}_I + (\bar{y}_I^2-\bar{z}_I^2)\frac{\Delta\bar{y}_J}{2}}{(\bar{y}_I+\frac{\Delta\bar{y}_J}{2})^2+\bar{z}_I^2} - \frac{(\bar{y}_I^2+\bar{z}_I^2)\bar{y}_I - (\bar{y}_I^2-\bar{z}_I^2)\frac{\Delta\bar{y}_J}{2}}{(\bar{y}_I-\frac{\Delta\bar{y}_J}{2})^2+\bar{z}_I^2} \\
L &= \bar{y}_I F + \frac{\bar{y}_I^2+\bar{z}_I^2+\bar{y}_I\frac{\Delta\bar{y}_J}{2}}{(\bar{y}_I+\frac{\Delta\bar{y}_J}{2})^2+\bar{z}_I^2} - \frac{\bar{y}_I^2+\bar{z}_I^2-\bar{y}_I\frac{\Delta\bar{y}_J}{2}}{(\bar{y}_I-\frac{\Delta\bar{y}_J}{2})^2+\bar{z}_I^2} \\
M &= F + \frac{\bar{y}_I+\frac{\Delta\bar{y}_J}{2}}{(\bar{y}_I+\frac{\Delta\bar{y}_J}{2})^2+\bar{z}_I^2} - \frac{\bar{y}_I-\frac{\Delta\bar{y}_J}{2}}{(\bar{y}_I-\frac{\Delta\bar{y}_J}{2})^2+\bar{z}_I^2}
\end{aligned}
$$

[8] state that equation 162 is only used for large values of $\bar{z}_I$, i.e. when the ε factor in equation 154 satisfies

$$\frac{1}{|\varepsilon|} \le 0.1 \text{ and } \frac{|\bar{z}_I|}{\Delta\bar{y}/2} > 0.001$$

The approximation

$$
\begin{aligned}
D_{2_{I,J}}(\omega) \approx \frac{\Delta\bar{x}_J\frac{\Delta\bar{y}_J}{2}}{\bar{y}_I^2+\bar{z}_I^2-\frac{\Delta\bar{y}_J^2}{4}} & \left[\frac{2(\bar{y}_I^2+\bar{z}_I^2+\frac{\Delta\bar{y}_J^2}{4})(\frac{\Delta\bar{y}_J^2}{4}A_2+C_2)+\bar{y}_I\Delta\bar{y}_J^2B_2}{\left((\bar{y}_I+\frac{\Delta\bar{y}_J}{2})^2+\bar{z}_I^2\right)\left((\bar{y}_I-\frac{\Delta\bar{y}_J}{2})^2+\bar{z}_I^2\right)}\right. \\
& \left. -\frac{4a}{\Delta\bar{y}_J^2}\left((\bar{y}_I^2+\bar{z}_I^2)A_2+\bar{y}_IB_2+C_2\right)\right] \qquad (163)
\end{aligned}
$$

is used for cases where

$$\frac{1}{|\varepsilon|} > 0.1 \text{ and } \frac{|\bar{z}_I|}{\Delta\bar{y}/2} > 0.001$$

where a is obtained by solving equation 155, such that

$$a = \frac{\Delta\bar{y}_J^2/4}{\bar{z}_I^2}\left(1-\frac{\bar{y}_I^2+\bar{z}_I^2-\Delta\bar{y}_J^2/4}{\Delta\bar{y}_J}F\right)$$

irrespective of which of equations 152 or 155 was used to calculate F. Note that [8] misquote equation 163; the equation given here is taken from [9]. Cases for which

$$\frac{|\bar{z}_I|}{\Delta\bar{y}/2} < 0.001$$

are assumed to be planar and $D_{2_{I,J}}(\omega) = 0$.

B.4 Normalization

The DLM is usually normalized such that it relates the normalized upwash

$$\bar{w}_I(\omega) = \frac{w_I(\omega)}{U_\infty}$$

to the pressure jump coefficient

$$\Delta c_{p_I}(\omega) = \frac{\Delta p_J(\omega)}{\frac{1}{2}\rho_\infty U_\infty^2}$$

Then, equation 38 becomes

$$\bar{w}_I(\omega) = -\frac{1}{8\pi}\sum_{J=1}^{N}\Delta c_{p_J}(\omega)\left(D_{1_{I,J}}(\omega) + D_{2_{I,J}}(\omega)\right) \tag{164}$$

The matrices $D_{1_{I,J}}$ and $D_{2_{I,J}}$ are functions of the wing geometry, panel discretization, ω and U_∞. It is desirable to render the matrices independent of U_∞ by introducing the characteristic reduced frequency

$$k = \frac{\omega b_{\text{char}}}{U_\infty} \tag{165}$$

where b_{char} is a characteristic chordwise length. Quite often it is defined as

$$b_{\text{char}} = \frac{\bar{\bar{c}}}{2}$$

where $\bar{\bar{c}}$ is the mean aerodynamic chord of the main wing. In order to introduce k in the expressions for $D_{1_{I,J}}$ and $D_{2_{I,J}}$, all lengths are normalized by b_{char}, such that

$$x' = \frac{x}{b_{\text{char}}},\ y' = \frac{y}{b_{\text{char}}},\ z' = \frac{z}{b_{\text{char}}},\ \lambda' = \frac{\lambda}{b_{\text{char}}},\ r' = \frac{r}{b_{\text{char}}}$$

both in global and in panel coordinates. From equations 125, 128 and 130, v, u and u_1 are all non-dimensional so that they are not affected by the normalization. The reduced frequency based on r inequation 127 becomes

$$k_1 = \frac{\omega r}{U_\infty} = \frac{\omega b_{\text{char}} r'}{U_\infty} = kr'$$

so that k_1 becomes a function of k only. Consequently, I_1, I_2, K_1, K_2 in equations 135, 137, 134 and 136 respectively, are also non-dimensional and functions of k only. Furthermore, T_1 in equation 132 is non-dimensional but T_2^* in equation 159 becomes

$$T_2^* = b_{\text{char}}^2\left(z_0'\cos\gamma - y_0'\sin\gamma\right)\left(z_0'\cos\gamma_s - y_0'\sin\gamma_s\right) = b_{\text{char}}^2 T_2^{*'}$$

Equation 149 becomes

$$\begin{aligned} D_{1_{I,J}}(k) &= b_{\text{char}}\Delta\bar{x}'_J \int_{-\Delta\bar{y}'_J/2}^{\Delta\bar{y}'_J/2} \frac{A'_1\bar{y}'^2 + B'_1\bar{y}' + C'_1}{b_{\text{char}}^2 r'(\bar{y}')^2} b_{\text{char}}\mathrm{d}\bar{y}' \\ &= \Delta\bar{x}'_J \int_{-\Delta\bar{y}'_J/2}^{\Delta\bar{y}'_J/2} \frac{A'_1\bar{y}'^2 + B'_1\bar{y}' + C'_1}{r'(\bar{y}')^2}\mathrm{d}\bar{y}' \end{aligned}$$

where

$$P_1(\bar{y}', k) = \mathrm{e}^{-\mathrm{i}kx_0'(\bar{y}')}K_1(\bar{y}', k)T_1 \approx A'_1\bar{y}'^2 + B'_1\bar{y}' + C_1$$

and

$$\begin{aligned} A'_1 &= \frac{P_1(-\Delta\bar{y}'_J/2) - 2P_1(0) + P_1(\Delta\bar{y}'_J/2)}{2(\Delta\bar{y}'_J/2)^2} \\ B'_1 &= \frac{P_1(\Delta\bar{y}'_J/2) - P_1(-\Delta\bar{y}'_J/2)}{\Delta\bar{y}'_J} \\ C'_1 &= P_1(0) \end{aligned}$$

such that

$$A_1 = \frac{A'_1}{b_{\text{char}}^2},\ B_1 = \frac{B'_1}{b_{\text{char}}},\ C_1 = C'_1$$

Finally, equation 152 becomes $F = F'/b_{\text{char}}$, where

$$F' = \frac{1}{|\bar{z}'_I|}\tan^{-1}\left(\frac{\Delta\bar{y}'_J|\bar{z}'_I|}{\bar{y}'^2_I + \bar{z}'^2_I - \Delta\bar{y}'^2_J/4}\right)$$

while G in equation 153 is non-dimensional. Substituting all this information into equation 151 leads to

$$D_{1_{I,J}}(k) = \Delta\bar{x}'_J\left[\left(\bar{y}'^2_I - \bar{z}_I{}'^2)A'_1 + \bar{y}'_I B'_1 + C_1\right)F' + \left(\frac{B'_1}{2} + \bar{y}'_I A'_1\right)G + \Delta\bar{y}'_J A'_1\right] \quad (166)$$

Equation 160 becomes

$$\begin{aligned} D_{2_{I,J}}(k) &= b_{\text{char}}\Delta\bar{x}'_J\int_{-\Delta\bar{y}'_J/2}^{\Delta\bar{y}'_J/2} \mathrm{e}^{-\mathrm{i}kx'_0(\bar{y}')}\frac{K_2(\bar{y},k)b_{\text{char}}^2 T_2^{*'}(\bar{y}')}{b_{\text{char}}^4 r'(\bar{y}')^4}b_{\text{char}}\mathrm{d}\bar{y}' \\ &= \Delta\bar{x}'_J\int_{-\Delta\bar{y}'_J/2}^{\Delta\bar{y}'_J/2}\frac{A'_2\bar{y}'^2 + B'_2\bar{y}' + C'_2}{r'(\bar{y}')^4}\mathrm{d}\bar{y}' \end{aligned}$$

where

$$P'_2(\bar{y}',k) = \mathrm{e}^{-\mathrm{i}kx'_0(\bar{y}')}K_2(\bar{y}',k)T_2^{*'}(\bar{y}') \approx A'_2\bar{y}'^2 + B'_2\bar{y}' + C_2$$

and

$$\begin{aligned} A'_2 &= \frac{P'_2(-\Delta\bar{y}'_J/2) - 2P'_2(0) + P'_2(\Delta\bar{y}'_J/2)}{2(\Delta\bar{y}'_J/2)^2} \\ B'_2 &= \frac{P'_2(\Delta\bar{y}'_J/2) - P'_2(-\Delta\bar{y}'_J/2)}{\Delta\bar{y}'_J} \\ C'_2 &= P'_2(0) \end{aligned}$$

such that

$$P_2 = b_{\text{char}}^2 P'_2,\ A_2 = A'_2,\ B_2 = b_{\text{char}}B'_2,\ C_2 = b_{\text{char}}^2 C'_2$$

Consequently, equation 162 becomes

$$D_{2_{I,J}}(\omega) = \frac{\Delta\bar{x}'_J}{2\bar{z}'^2_I}\left(K'A'_2 + L'B'_2 + M'C'_2\right) \quad (167)$$

where

$$\begin{aligned} K' &= \left(\bar{y}'^2_I + \bar{z}_I{}'^2\right)F' + \frac{(\bar{y}_I{}'^2 + \bar{z}'^2_I)\bar{y}'_I + (\bar{y}'^2_I - \bar{z}'^2_I)\frac{\Delta\bar{y}'_J}{2}}{(\bar{y}'_I + \frac{\Delta\bar{y}'_J}{2})^2 + \bar{z}'^2_I} - \frac{(\bar{y}'^2_I + \bar{z}'^2_I)\bar{y}'_I - (\bar{y}'^2_I - \bar{z}'^2_I)\frac{\Delta\bar{y}'_J}{2}}{(\bar{y}'_I - \frac{\Delta\bar{y}'_J}{2})^2 + \bar{z}'^2_I} \\ L' &= \bar{y}'_I F' + \frac{\bar{y}'^2_I + \bar{z}'^2_I + \bar{y}'_I\frac{\Delta\bar{y}'_J}{2}}{(\bar{y}'_I + \frac{\Delta\bar{y}'_J}{2})^2 + \bar{z}'^2_I} - \frac{\bar{y}'^2_I + \bar{z}'^2_I - \bar{y}'_I\frac{\Delta\bar{y}'_J}{2}}{(\bar{y}'_I - \frac{\Delta\bar{y}'_J}{2})^2 + \bar{z}'^2_I} \\ M' &= F' + \frac{\bar{y}'_I + \frac{\Delta\bar{y}'_J}{2}}{(\bar{y}'_I + \frac{\Delta\bar{y}'_J}{2})^2 + \bar{z}'^2_I} - \frac{\bar{y}'_I - \frac{\Delta\bar{y}'_J}{2}}{(\bar{y}'_I - \frac{\Delta\bar{y}'_J}{2})^2 + \bar{z}'^2_I} \end{aligned}$$

such that

$$K = b_{\text{char}}K',\ \ L = L',\ \ M = \frac{1}{b_{\text{char}}}M'$$

Consequently, the equations for D_1 and D_2 are identical in form, whether written in terms of dimensional or normalized coordinates. The only differences lie in the exponential terms, where $\omega x_0/U_\infty$ becomes kx_0' and $\omega r/U_\infty$ becomes kr'. The final normalized form of equation 164 is given by

$$\bar{w}_I(k) = -\frac{1}{8\pi}\sum_{J=1}^{N}\Delta c_{p_J}(k)\left(D_{1_{I,J}}(k) + D_{2_{I,J}}(k)\right) \tag{168}$$

where $\bar{w}_I(k)$ is the normalized upwash written in terms of the characteristic reduced frequency.