〔研究ノート〕

NBU GEN−HEL の開発（第２報）
― NASA ロータ風洞試験とのロータ揚力／抗力比較検証 ―

大城　鳳花*，中山　周一*２

*日本文理大学工学部航空宇宙工学科（2024年度卒業）
*２日本文理大学工学部航空宇宙工学科

Helicopter Rotor Simulation Program NBU GEN−HEL Development（２nd Report）
― Verification Challenge to Compare with NASA Rotor Wind Tunnel Test Lift and Drag ―

Fuka OSHIRO*, Shuichi NAKAYAMA*２

*Department of Aerospace Engineering, School of Engineering, Nippon Bunri University（Graduate, AY2024）
*２Department of Aerospace Engineering, School of Engineering, Nippon Bunri University

１．はじめに

NASA CR（Contractor Report）として基本モデルが公開されている GEN−HEL[1]を本学で内製することを目標に2023年度から卒業研究で取り組み，2024年度には NASA によるロータ風洞試験結果[2]と比較検証を行いブレード失速による非線形性を捉える等の成果を得た。本稿では，基本的な計算確認を行った第１報[3]に引き続き，NASA ロータ風洞試験結果との比較検証のうち，ロータ揚力とロータ抗力の検証結果ついて報告する。

２．ブレード運動の数学モデル

ヘリコプタ・ブレードは，回転自由度を有するヒンジを介してハブに固定されており，ブレード回転に伴いヒンジ回りのフラップ角ζ，リードラグ角ζが変化する。このフラップ角，リードラグ角に関する運動方程式は次式のように表される[3]。

$$\begin{bmatrix} mr_{cg}^2 & 0 \\ 0 & mr_{cg}^2 \end{bmatrix} \begin{Bmatrix} \ddot{\zeta} \\ \ddot{\beta} \end{Bmatrix} + \begin{bmatrix} 0.05\left(2mr_{cg}\Omega\sqrt{r_{cg}e}\right) & 0 \\ 0 & 0 \end{bmatrix} \begin{Bmatrix} \dot{\zeta} \\ \dot{\beta} \end{Bmatrix} + \begin{bmatrix} mer_{cg}\Omega^2 & 0 \\ 0 & mr_{cg}\left(e+r_{cg}\right)\Omega^2 \end{bmatrix} \begin{Bmatrix} \zeta \\ \beta \end{Bmatrix} = r_a \begin{Bmatrix} Fb_X \\ -Fb_Z \end{Bmatrix} \quad (1)$$

ここで，mはブレード質量，r_{cg}はブレード重心とヒンジまでの距離，eはハブ中心からヒンジまでの距離，Ωはロータ回転角速度である。右辺は次式で表されるブレード空気力であり，r_aはヒンジからブレード空力代表点までの距離である[3]。

$$\begin{Bmatrix} Fb_X \\ Fb_Y \\ Fb_Z \end{Bmatrix} = \frac{1}{u_0 u_{tr}} \begin{bmatrix} u_t u_{tr} & u_r u_0 & -u_t u_p \\ -u_r u_{tr} & u_t u_0 & u_r u_p \\ u_p u_{tr} & 0 & u_{tr}^2 \end{bmatrix} \begin{Bmatrix} -d \\ 0 \\ -l \end{Bmatrix} \tag{2}$$

$\{Fb_X, Fb_Y, Fb_Z\}^T$は，ブレード相対風座標系で定義されるブレード揚力lとブレード抗力dをブレード固定座標系に変換したもので，両座標系の変換行列は，ブレード固定座標系で表したブレード相対風$\{u_t, -u_r, u_p\}^T$および同速度から構成される$u_{tr} = \sqrt{u_t^2 + u_r^2}$, $u_0 = \sqrt{u_t^2 + u_r^2 + u_p^2}$で表現される。ブレード相対風$\{u_t, -u_r, u_p\}^T$は，次式により，ブレードフラップ角，リードラグ角の他，飛行速度V，ロータシャフト迎え角α_sで表される。付録に収録するソースコードでは，$\cos\zeta \simeq 1$，$\sin\zeta \simeq \zeta$，$\cos\beta \simeq 1$，$\sin\beta \simeq \beta$と近似し変数同士の積を省略している(3)。

$$\begin{Bmatrix} u_t \\ -u_r \\ u_p \end{Bmatrix} = e \begin{Bmatrix} \Omega\cos\zeta \\ \Omega\sin\zeta\cos\beta \\ \Omega\sin\zeta\sin\beta \end{Bmatrix} + r_{cg} \begin{Bmatrix} (\Omega + \dot{\zeta})\cos\beta \\ 0 \\ -\dot{\beta} \end{Bmatrix} + V \begin{Bmatrix} \cos\alpha_s \sin(\psi + \zeta) \\ -\cos\alpha_s \cos(\psi + \zeta)\cos\beta - \sin\alpha_s \sin\beta \\ -\cos\alpha_s \cos(\psi + \zeta)\sin\beta + \sin\alpha_s \cos\beta \end{Bmatrix} \tag{3}$$

3．NASA ロータ風洞試験

本稿の数値計算結果はNASA−TM4183記載の風洞試験データ(2)と比較検証する。TM4183ではBaselineブレードとしてCR166309が対象としているUH−60のロータ風洞試験が行われている。TM4183からBaselineブレードの試験条件を表1に示す。TM4183のデータはNASA LangleyのTDT（Transonic Dynamic Tunnel）で取得された。TDTは風洞全体を加圧することで実機レイノルズ数に近づけた風洞試験が可能で，表1から海面上標準大気の空気密度1.225［kg/m^3］=0.0023769［slug/ft^3］よりも高い気体密度で風洞試験が行われている。さらにTM4183のデータが取得された1990年当時はR−12フロンガスが使用されていた。R−12の粘性係数はおよそ12.5［μPa−s］(4)であり，空気の18［μPa−s］（海面上標準大気）よりも低くレイノルズ向上効果がある。

表1の多くの条件でM_T=0.65となっているのは，マッハ数が実機に一致していることを示す。後掲する表3からUH−60実機のホバリング時ブレード翼端速度は221［m/s］であり，海面上標準大気の15℃での翼端マッハ数は0.65である。以上から，風洞試験のレイノルズ数をRe_{WTT}, UH−60実機のレイノルズ数をRe_{WTT}と表すと，風洞試験のレイノルズ数比率「%Re」は次式となる。

$$\%\mathrm{Re} = \frac{Re_{WTT}}{Re_{UH60}} = \frac{\rho_{WTT} V_{WTT} L_{WTT}}{\mu_{WTT}} \frac{\mu_{UH60}}{\rho_{UH60} V_{UH60} L_{UH60}}$$

$$= \left(\frac{\rho_{WTT}}{\rho_{UH60}}\right)\left(\frac{V_{WTT}}{V_{UH60}}\right)\left(\frac{L_{WTT}}{L_{UH60}}\right)\left(\frac{\mu_{UH60}}{\mu_{WTT}}\right)$$

M_T=0.65の条件では$\left(\frac{V_{WTT}}{V_{UH60}}\right)$は1.0，$\left(\frac{L_{WTT}}{L_{UH60}}\right)$は模型縮率で1／6，速度比はR−12フロンの粘性を反映し，風洞試験のレイノルズ数の実機比「%Re」を計算し表1に示している。

$$\%\mathrm{Re} = \frac{Re_{WTT}}{Re_{UH60}} = \left(\frac{\rho_{WTT}}{\rho_{UH60}}\right)\left(\frac{1}{6}\right)\left(\frac{18}{12.5}\right)$$

表1にあるγ_bは，ブレードに働く慣性力と空気力の比率を示すロック数と呼ばれる無次元数で(5)，Rはロータ半径，cはブレード翼弦長，ρは空気密度，$a(=2\pi)$は揚力傾斜，I_bはブレードの慣性モーメントである。

$$\gamma_b = \frac{\rho a c R^4}{I_b} = \frac{\rho a c R^4}{m r_{cg}^2} \tag{4}$$

表3に後掲するUH−60実機諸元からロック数を計算すると8.9となり，表1の試験条件の中では9.35に近い。以上から，7（c）を検証条件として採用する。

7（c）の試験結果の一部を表2に示す。α_s，A_1，B_1，θ_0，μが試験条件でC_L，C_D，C_Qが6分力天秤による計測結果である。α_sはロータシャフトの迎角，μはアドバンス比と呼ばれるホバリング時のブレード翼端速度ΩRと風洞風速（飛行速度）の比である(5)。

$$\mu = \frac{V\cos\alpha_s}{\Omega R} \tag{5}$$

A_1，B_1，θ_0は次式によりブレードピッチ角θを構成し，θ_0はコレクティブピッチ角，A_1，B_1はサイクリックピッチ角と呼ばれる(5)。

$$\theta = \theta_0 - A_1\cos\psi - B_1\sin\psi \tag{6}$$

C_L，C_D，C_Qは風洞風を基準とする慣性座標系でのロータ合力の無次元化数であり，(2)式のブレード固定座標系でのブレード空気力を座標変換行列(3)

$[\alpha_s]_{I/HF}, [\psi_j]_{HF/HR}, [\zeta_j]_{HR/BW}, [\beta_j]_{BW/B}$ により変換し，

$$\begin{Bmatrix} -D_j \\ Y_j \\ -L_j \end{Bmatrix} = [\alpha_s]_{I/HF}[\psi_j]_{HF/HR}[\zeta_j]_{HR/BW}[\beta_j]_{BW/B} \begin{Bmatrix} Fb_{Xj} \\ Fb_{Yj} \\ Fb_{Zj} \end{Bmatrix} \tag{7}$$

j= 0，1，2，3の各ブレード分を合算してロータ揚力 L，ロータ抗力 D を求める。

$$D = \sum_{j=0}^{3} D_j \, , L = \sum_{j=0}^{3} L_j \tag{8}$$

以上をロータの円面積，ホバリング時の翼端速度により無次元化し C_L，C_D を求める[2,5]。

$$\begin{cases} C_L = \dfrac{L}{\rho\pi R^2(\Omega R)^2} \\ C_D = \dfrac{D}{\rho\pi R^2(\Omega R)^2} \end{cases} \tag{9}$$

表１　NASA ロータ風洞試験条件[2]

NASA TM-4183						%Re	検証相手
Blade set	ρ slug/ft³	$I_b$slug-ft²	M_T	γ_b	Table		
Baseline (0.00469)	0.0023	0.4383	0.28	4.58	7(a)	0.24	
	0.00382		0.65	7.61	7(b)	0.40	
	0.00469		0.65	9.35	7(c)	0.49	✓
	0.006		0.65	11.95	7(d)	0.63	
	0.0075		0.65	14.94	7(e)	0.78	
	0.009		0.65	17.93	7(f)	0.94	
Baseline (0.006)	0.006	0.5602	0.628	9.35	8(a)	0.63	
	0.006		0.65	9.35	8(b)	0.63	
Baseline (0.0076)	0.0076	0.7092	0.628	9.35	9(a)	0.79	
	0.0076		0.65	9.35	9(b)	0.79	

表２　NASA ロータ風洞結果例（Table７(c）の代表例)[2]

POINT	α_S	A_1	B_1	θ_0	μ	$M_{1,90}$	C_L	C_D	C_Q
1135	−5.2	−1.7	2.7	4	0.251	0.819	0.00246	−0.00009	0.00017
1136	−5.2	−2.5	3.4	6	0.252	0.820	0.00392	−0.00025	0.00023
1137	−5.2	−3.4	4	8	0.25	0.819	0.00536	−0.00041	0.00031
1139	−5.2	−4.2	4.7	10	0.25	0.819	0.00677	−0.00056	0.00041

４．計算結果

４－１　計算諸元およびプログラム

watlab-blog.com[6]を参考に(1)式を odeint により時間積分する python プログラムを作成した。odeint は運動方程式の変数（ζ,β）を戻り値として返すが，ブレード空気力等の変数を引っ張り出す必要があり，その方法は stackoverflow.com[7]を参考にした。表１にあるように検証データは実機レイノルズに近い条件で取得されているので，計算諸元は表３に示す実機諸元とする。ソースコードを付録に収録する。

４－２　NASA との比較検証（その１）

NASA 試　験[2]の μ =0.25，α_s = −5.2［deg］，θ_0 =12［deg］，A_1 = −5.2［deg］，B_1 =5.4［deg］で計算した時歴を図１に示す。

表３　計算（実機）諸元[3]

記号	値	意味
Ω	27[rad/s]	ロータ回転角速度
m	72.5[kg]	ブレード質量
R	8.178[m]	ロータ半径
e	0.381[m]	ヒンジオフセット
r_{cg}	5.32[m]	ブレード重心位置
r_a	4.7[m]	ブレード空力半径
r_Q	6.1[m]	トルク半径
S	3.36[m²]	ブレード面積
c	0.527[m]	ブレード翼弦長
ρ	1.225[kg/m³]	標準大気

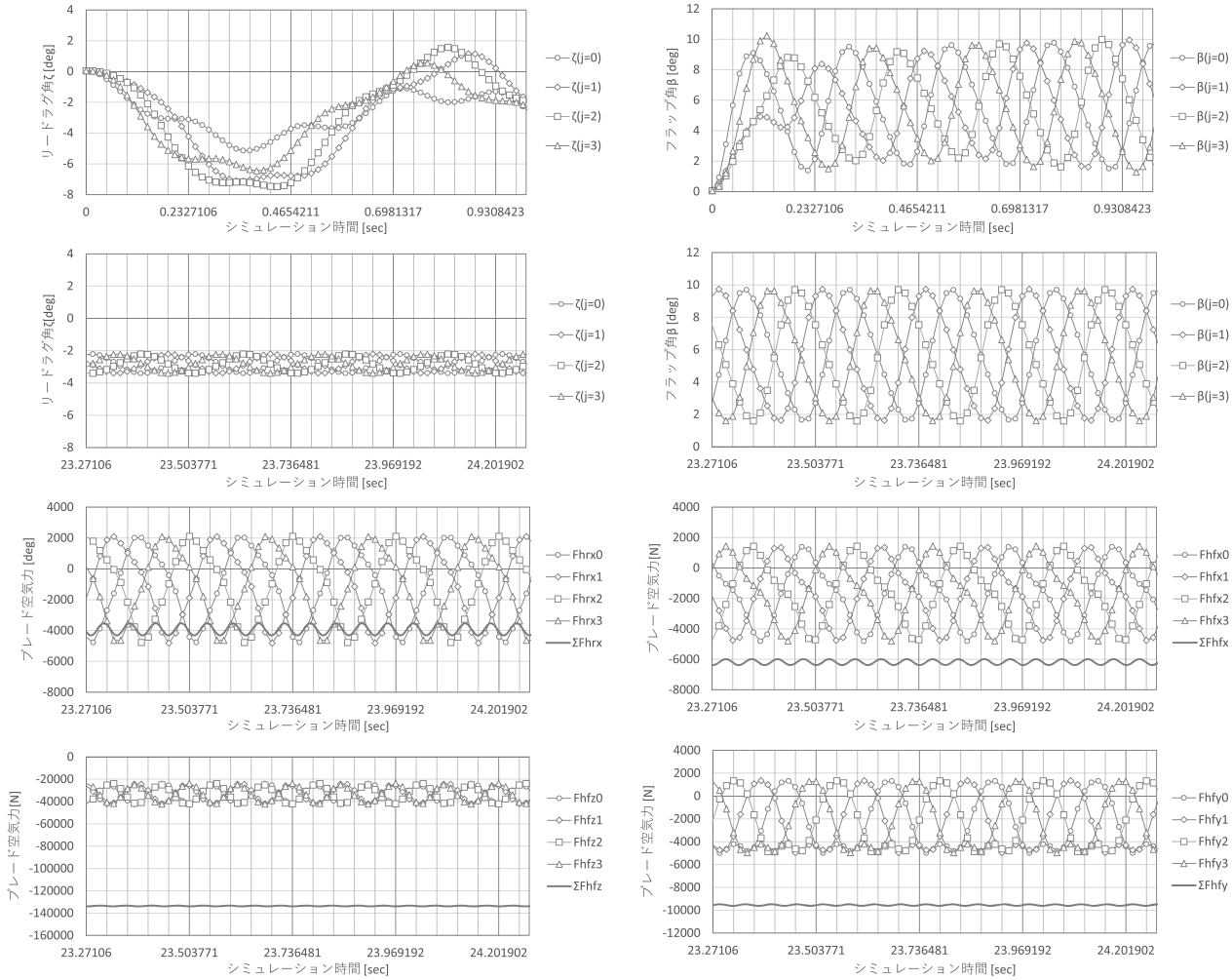

図１　検証結果（その１）時歴データ

第１報[3]の図11と同様にj= 0ブレード○は機体前方位置（主目盛線）でβが最大，機体後方（真ん中の補助目盛線位置）で最小となっており，ロータが後傾していることがわかる。β，ζがロータ回転に応じた周期解となっているように，ハブ回転座標系[3]のFhr_{Xj}，ハブ固定座標系[3]のFhf_{Z_J}等のブレード空気力にもロータ回転に合わせた周期変動がみられる。

同様な計算を行い，図１以外のθ_0，A_1，B_1の結果も併せて同じμ，αの結果をまとめて，縦軸：揚力係数，横軸：抗力係数でグラフ化した結果を図２に示す。

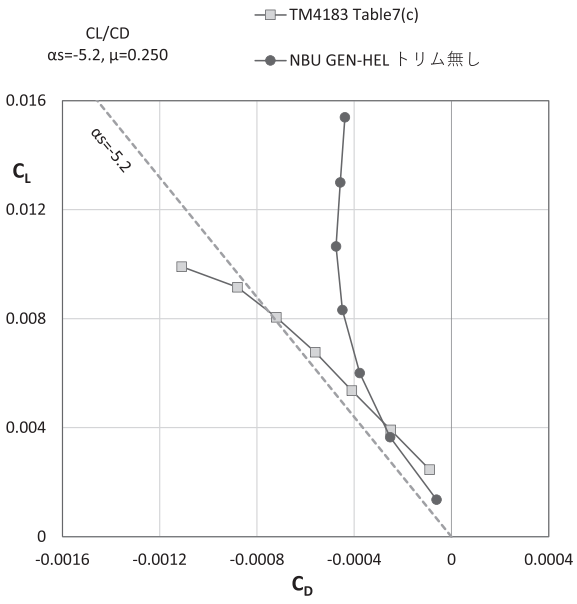

図２　NASA 試験条件での計算結果

４－３　NASA との比較検証（その２）

図２においてNASA 試験結果はシャフト迎角方向に概ね沿っているのに対し計算結果は大きくずれている。TM4183に「At each collective pitch setting, the cyclic pitch was used to remove rotor first harmonic flapping with respect to the rotor shaft.」とあり，B_1，A_1の設

定方法が述べられている。B_1，A_1を TM4183値に合わせるのではなく，NBU Gen−Hel でB_1，A_1を同じように設定する方法を検討する。計算されたフラップ角βを次式で表した際に，前記引用部は$a_1 = 0$，$b_1 = 0$を意味するので，これを達成するB_1，A_1を繰り返し計算で求める。

$$\beta = a_0 + a_1\cos\psi + b_1\sin\psi + a_2\cos 2\psi + b_2\sin 2\psi \cdots, \ (\psi = \Omega t) \tag{10}$$

いま繰り返し計算の途中 i 番目とし，$B_{1,i}$，$A_{1,i}$に対する計算結果を$a_{1,i}(\neq 0)$，$b_{1,i}(\neq 0)$と表す。（i＋１）番目の状態は次式で表すことができる。

$$\begin{cases} a_{1,i+1} = a_{1,i} + \left(\dfrac{\partial a_1}{\partial B_1}\right)_i \left(B_{1,i+1} - B_{1,i}\right) + \left(\dfrac{\partial a_1}{\partial A_{1S}}\right)_i \left(A_{1,i+1} - A_{1,i}\right) \\ b_{1,i+1} = b_{1,i} + \left(\dfrac{\partial b_1}{\partial B_1}\right)_i \left(B_{1,i+1} - B_{1,i}\right) + \left(\dfrac{\partial b_1}{\partial A_{1S}}\right)_i \left(A_{1,i+1} - A_{1,i}\right) \end{cases} \tag{11}$$

行列形式に書き直すと次となる。$[\Phi_i]$はヤコビ行列である。

$$\begin{Bmatrix} a_{1,i+1} \\ b_{1,i+1} \end{Bmatrix} = \begin{Bmatrix} a_{1,i} \\ b_{1,i} \end{Bmatrix} + \begin{bmatrix} \left(\dfrac{\partial a_1}{\partial B_1}\right)_i & \left(\dfrac{\partial a_1}{\partial B_1}\right)_i \\ \left(\dfrac{\partial b_1}{\partial B_1}\right)_i & \left(\dfrac{\partial b_1}{\partial A_1}\right)_i \end{bmatrix} \begin{Bmatrix} B_{1,i+1} - B_{1,i} \\ A_{1,i+1} - A_{1,i} \end{Bmatrix} = \begin{Bmatrix} a_{1,i} \\ b_{1,i} \end{Bmatrix} + [\Phi_i] \begin{Bmatrix} B_{1,i+1} - B_{1,i} \\ A_{1,i+1} - A_{1,i} \end{Bmatrix} \tag{12}$$

操舵量について，$a_{1,i}$，$b_{1,i}$をゼロに近づけるための修正量を$\delta B_{1,i}$，$\delta A_{1,i}$で表し，

$$\begin{cases} B_{1,i+1} = B_{1,i} + \delta B_{1,i} \\ A_{1,i+1} = A_{1,i} + \delta A_{1,i} \end{cases} \tag{13}$$

繰り返し計算の目標として$a_{1,i+1} = b_{1,i+1} = 0$であることを踏まえると

$$\begin{Bmatrix} \delta B_{1,i} \\ \delta A_{1,i} \end{Bmatrix} = -[\Phi_{\mathrm{i}}]^{-1} \begin{Bmatrix} a_{1,i} \\ b_{1,i} \end{Bmatrix} \tag{14}$$

となる。線形であれば，これで操舵修正量を求めることができるが，本問題は非線形なので⑭式で得られた操舵修正量に対して1.0より小さい係数（付録に示すプログラムでは0.3）を乗じて操舵量の修正を行う操作を繰り返し，$a_1 \approx b_1 \approx 0$に近づいたかの収束判定を行う。ヤコビ行列$[\Phi_i]$は次式により数値計算で求める。

$$[\Phi_{\mathrm{i}}] = \begin{bmatrix} \dfrac{a_1(B_{1,i} + \delta, A_{1,i}) - a_1(B_{1,i}, A_{1,i})}{\delta} & \dfrac{a_1(B_{1,i}, A_{1,i} + \delta) - a_1(B_{1,i}, A_{1,i})}{\delta} \\ \dfrac{b_1(B_{1,i} + \delta, A_{1,i}) - b_1(B_{1,i}, A_{1,i})}{\delta} & \dfrac{b_1(B_{1,i}, A_{1,i} + \delta) - b_1(B_{1,i}, A_{1,i})}{\delta} \end{bmatrix} \tag{15}$$

以上の繰り返し計算により$a_1 \approx b_1 \approx 0$となる$B_1$，$A_1$を求めた場合の計算結果を図３に示す。図２と異なり，フラップ角から１Ωの周期変動が無くなっていることが確認できる。Fhr_{Zj}などのブレード空気力を見ると，振幅は図２とほぼ同じレベルであるが，その中心値はゼロに近づき，j= 0，1，2，3の和は小さくなっている。図３以外のθ_0，の結果も併せて同じμ，α_sの結果をまとめてグラフ化した結果を図４に示す。

a_1，b_1をゼロに近づけた▲は NASA 試験結果に近くなった。トリムの方法として，シャフトに対してロータ面を直交させることからC_L，C_Dのラインはシャフト迎角の方向を向くことが確認できる。なお，ロータ揚力が高い場合，⑭式の繰り返し計算の収束が難しい場合がある。

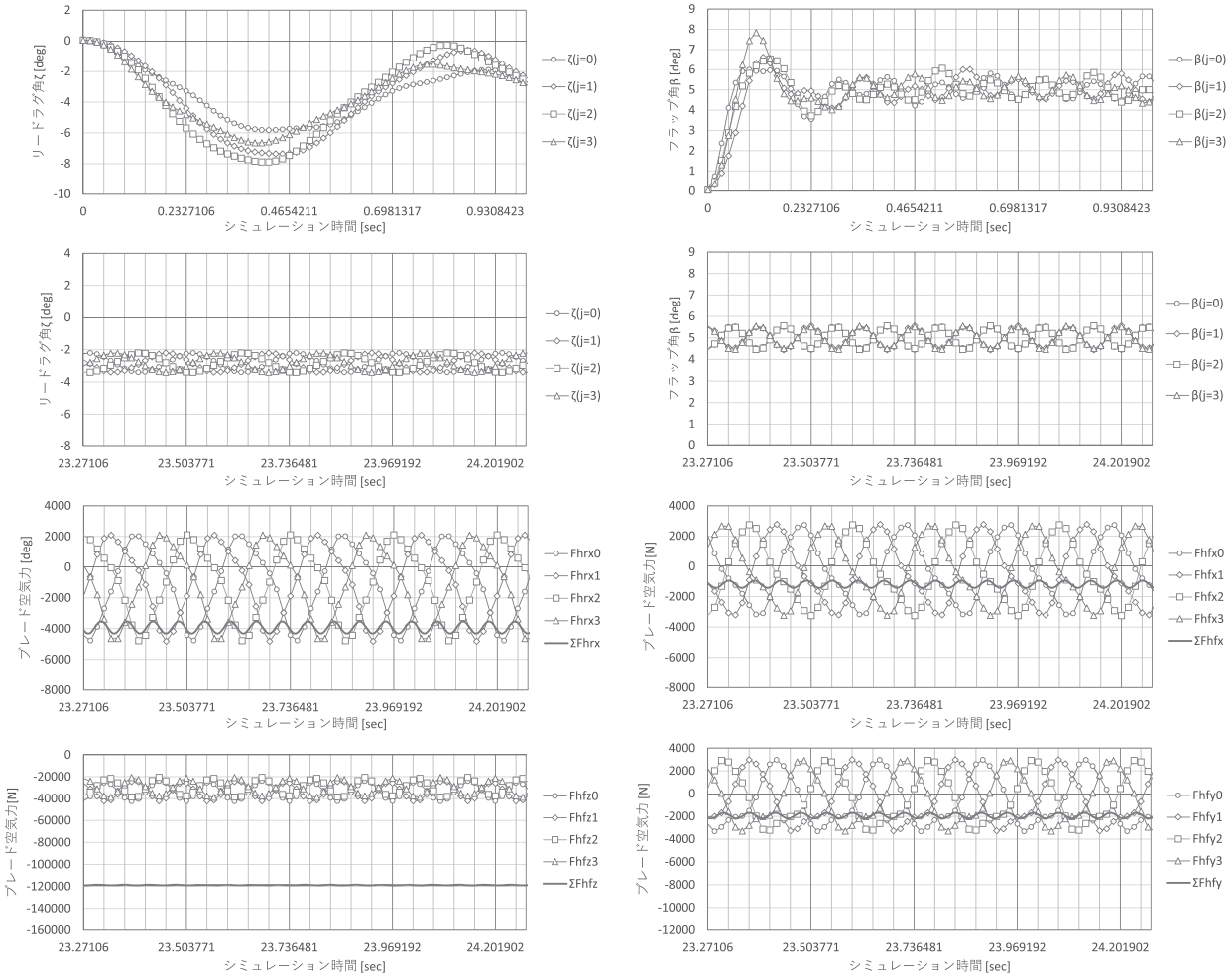

図３　NASA 比較検証結果（その２）時歴データ

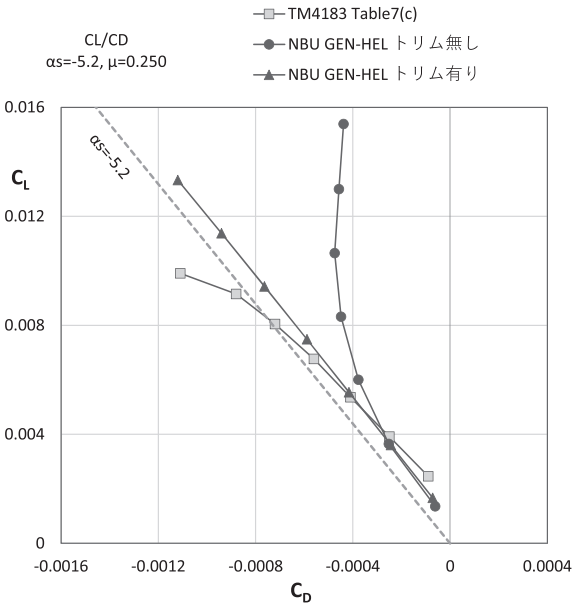

図４　検証結果（その２）揚抗特性

４－４　NASA との比較検証（その３）

ただし，NASA 風洞試験結果は，揚力係数0.008以上では合力が前に垂れてきている。この要因としてブレードが失速に近づいていることが考えられる。

４－３までの計算では第１報(28)式に示すように，ブレード断面の揚力係数は迎角に比例，抗力係数は一定と仮定しており，失速は考慮されていない。

一方 CR には UH-60ブレードで使われている SC1095翼の揚力／抗力データが掲載されている。図５に示すように高い迎角では失速と思われる揚力の増加の頭打ち，抗力増を示している。そこで，失速の影響を評価するため，第１報の(28)式に代わり CR[(1)]のc_l，c_dテーブルの M＝0.4のデータに線形補間して使用する計算を行った。0.01で一定と仮設定していた抗力係数についても同様な補間を行った。その結果を図６に■で示す。図６から，ロータ揚力係数0.008以上の領域でロータ合力が前垂れする様子を NBU Gen-Hel は捉えており，前垂れの要因はブレード失速である可能性が高いことが分かった。

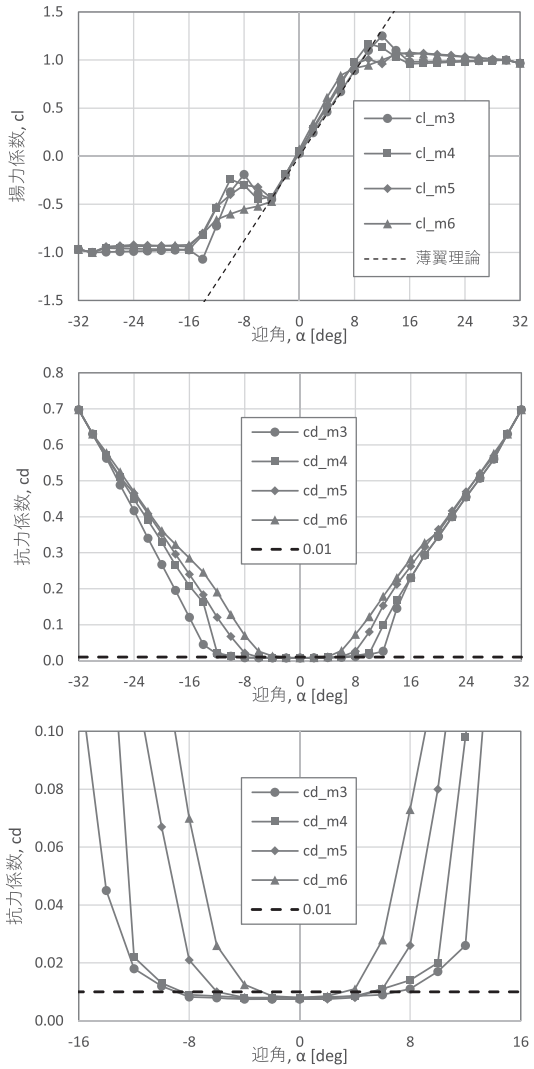

図５　２次元翼テーブルデータ

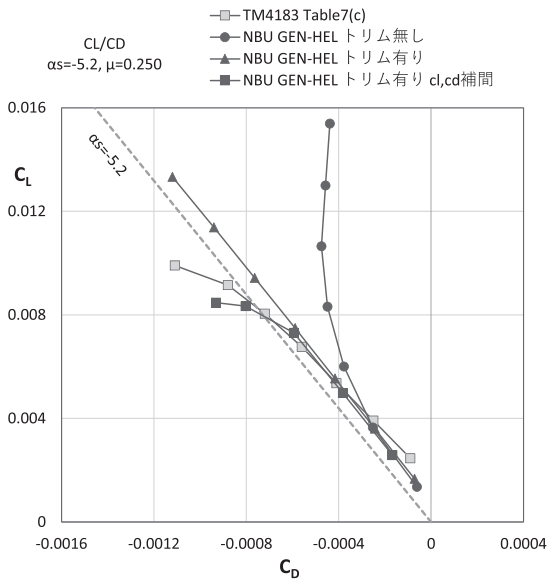

図６　検証結果（その２）揚抗特性

図５の２次元翼型データを使うことで，図６に示されるようにロータのＸ軸方向の力が増え合力が前垂れする理由について考察する。参考文献（８）にあるように，ヘリコプタが前進飛行時に機体飛行速度とブレード回転速度が反対向きとなる後退側で迎角が大きくなる。CR が対象としている UH−60と同じ上方からみて反時計回りにロータが回転するヘリコプタの場合，ブレードの失速は主に左舷側で発生することになる。左舷側でのブレード失速は，ブレード相対風座標系でのブレード抗力を増大させるが，その方向をハブ固定座標系から見ると機体前方方向となる。このため，ブレード失速によりＸ軸方向の力が増えることになる。

５．おわりに

公開されている NASA CR(1)に基づいたシミュレーションプログラムを開発し，NASA TM(2)にある実機レイノルズ数のロータ風洞試験結果と比較検証した。ロータ６分力のうちロータ揚力とロータ抗力の関係について，失速効果を反映することで，ブレードの失速による高揚力域での非線形性を捉えることができた。

付録に収録するプログラムについて，2024年度卒業研究以降の課題を列挙しておく。

- ＦとＦ２で同じ計算が重複してコーディングされており，信頼性に難があるのを改善する。
- 第１報(41)式r_a等の間違いを修正する。
- 本稿では４－３を除き50秒間計算させているが，途中で収束判定をすて計算を止める。本稿に示したフラッピングの１Ω成分の導出するアルゴリズムと似たようなものになるはず。

参考文献

（１）Howlett, J. J., UH−60A Black Hawk engineering simulation program. Volume１：Mathematical model, NASA−CR－166309（1981）

（２）Singleton, Jeffrey D.Yeager, William T., Jr., Wilbur, Matthew L., Performance Data from a Wind -Tunnel Test of Two Main-rotor Blade Designs for a Utility-Class Helicopter, NASA−TM−4183（1990）

（３）大城鳳花，中山周一，NBU　GEN−HEL の開発（第１報），日本文理大学紀要，第53巻第２号

（４）高橋満男，高橋信次，岩崎廣次，気体ジクロロジフルオロメタン（R−12）の粘性率，化学工学論

文集／10巻１号（1984）
（５）加藤寛一郎，今永勇生，ヘリコプタ入門，東京大学出版会（1985）
（６）https://watlab-blog.com/2023/02/25/scipy-odeint/（2025年６月16日閲覧）
（７）https://stackoverflow.com/questions/74143515/how-to-return-ordinary-variables-from-scipy-odein（2025年６月16日閲覧）
（８）航空技術協会編集，図解ヘリコプタ入門，航空技術協会（1987）

（2025年６月16日受理）

付録

```
# NBU Gen-Hel 20250614
# programmed by Fuka Oshiro    2025.6.14

import numpy as np
from scipy.integrate import odeint
from scipy import interpolate
import datetime
import time

def model(omg, r_cg, e): # define model

    m = 72.5 #kg

    M = np.array([[m * r_cg**2, 0],                                                                    第1報(15)式
                  [0, m * r_cg**2]])                                                                   第1報(15)式

    D = np.array([[0.05 * (2 * m * r_cg * omg * np.sqrt(r_cg * e)), 0],                                第1報(33)式
                  [                                              0, 0]])                               第1報(33)式

    K = np.array([[m * e * r_cg * omg**2,          0],                                                 第1報(15)式
                  [0, m * r_cg * (e + r_cg) * omg**2]])                                                第1報(15)式

    return M, D, K

def f(var, t, M, D, K, omg, r_a, e, Vc, Vs, j, th0, A1s, B1s, S, rho, aero): # harmonic oscillator

    psi = omg * t + np.pi * j / 2

    th = th0 - A1s * np.cos(psi) - B1s * np.sin(psi) #rad                                              (6)式

    ut =  ((e + r_a) * omg + r_a * var[2]) + ( Vc * np.sin(psi) + var[0] * Vc * np.cos(psi))           (3)式
    ur = -((             e * omg * var[0]) + (-Vc * np.cos(psi) + var[0] * Vc * np.sin(psi)) - Vs * var[1])   (3)式
    up =  (                -r_a * var[3]) + (                  - var[1] * Vc * np.cos(psi)  + Vs)      (3)式

    u2tr = ut*ut + ur*ur
    u2trp = ut*ut + ur*ur + up*up
    utr = np.sqrt(u2tr)
    utrp = np.sqrt(u2trp)

    al = np.arctan((ut * np.tan(th) + up) / (utr - up * ut * np.tan(th) / utr)) #rad                   第1報(29)式

    if aero == 1 :
        al_deg = al * 180 / np.pi #deg
        if al_deg >= -32 and al_deg <= 32:
            cl = fitted_curve_cl(al_deg)
            cd = fitted_curve_cd(al_deg)
        else :
            cl = fitted_curve_cl_HighAngle(al_deg)
            cd = fitted_curve_cd_HighAngle(al_deg)
    else :
        cl = 5.39 * al                                                                                 第1報(28),(30)式
        cd = 0.01

    # ブレード相対風座標系からブレード固定座標系に変換
    WtoB = np.array([[  ut * utr, ur * utrp, - ut * up],                                               (2)式
                     [- ur * utr, ut * utrp,   ur * up],                                               (2)式
                     [  up * utr,         0,      u2tr]]                                               (2)式
                     / (utrp * utr))                                                                   (2)式

    # (ブレード座標系)
    lift = 0.5 * rho * u2trp * S * cl                                                                  第1報(22)式
    drag = 0.5 * rho * u2trp * S * cd                                                                  第1報(22)式
    Fw = np.array([-drag, 0, -lift])                                                                   第1報(22)式

    # 外力
    Fb = WtoB @ Fw                                                                                     (2)式

    x = var[0:int(len(var) / 2)]
    v = var[int(len(var) / 2)::]

    Mb = r_a * np.array([Fb[0], -Fb[2]])                                                               第1報(31)式

    # 質量行列の逆行列
    M_inv = np.linalg.inv(M)

    # 運動方程式
    a = - (M_inv @ K) @ x - (M_inv @ D) @ v + (M_inv @ Mb)

    output = np.concatenate([v, a])

    return output

def f2(var, t, omg, r_a, e, Vc, Vs, j, th0, A1s, B1s, S, rho, aero): # harmonic oscillator
                                                    (84～125行は、26～67行と同じ)
    # ブレード固定座標系からハブブレード中間座標系に変換
    BtoHB = np.array([[1,                0,                0],                                     第1報(3)式の逆
                      [0,  np.cos(var[1]), np.sin(var[1])],                                          第1報(3)式の逆
                      [0, -np.sin(var[1]), np.cos(var[1])]])                                         第1報(3)式の逆

    Fhb = BtoHB @ Fb

    # ハブブレード中間座標系からハブ回転座標系に変換
    HBtoHR = np.array([[ np.cos(var[0]), np.sin(var[0]), 0],                                           第1報(6)式の逆
                       [-np.sin(var[0]), np.cos(var[0]), 0],                                           第1報(6)式の逆
                       [              0,              0, 1]])                                          第1報(6)式の逆

    # ブレード1枚あたりの空気力（ハブ回転座標系）
    # トルクQの計算で使う
    Fhr = HBtoHR @ Fhb

```

```
    # ハブ回転座標系からハブ非回転座標系に変換
    HRtoHF = np.array([[np.sin(psi), -np.cos(psi), 0],                                                    第1報(9)式の逆
                       [np.cos(psi),  np.sin(psi), 0],                                                    第1報(9)式の逆
                       [     0     ,       0      , 1]])                                                 第1報(9)式の逆

    # ブレード1枚当たりの空気力（ハブ非回転座標系）
    Fhf = HRtoHF @ Fhr

    return Fw, Fb, Fhb, Fhr, Fhf

def Beta_Coef():

    row = 12772
    row1 = row + count
    X1 = X[row:row1, :] # [1, cos(psi), sin(psi)]
    y = var[row:row1, 1] # j1のβを1周分スライス

    #BetaCoef = [const, beta1_c, beta1_s]
    BetaCoef0 = np.linalg.pinv(X1.T @ X1) @ X1.T @ y
    BetaCoef0 = BetaCoef0.reshape(1, 3)

    row = 12832
    row1 = row + count
    X1 = X[row:row1, :] # [1, cos(psi), sin(psi)]
    y = var[row:row1, 1] # j1のβを1周分スライス

    #BetaCoef = [const, beta1_c, beta1_s]
    BetaCoef1 = np.linalg.pinv(X1.T @ X1) @ X1.T @ y
    BetaCoef1 = BetaCoef1.reshape(1, 3)

    d_BetaCoef = abs((BetaCoef1 - BetaCoef0) * 180 / np.pi) # deg

    if np.any(d_BetaCoef > 0.1): # deg 収束判定
        print('Betaが収束していません')
    # else:
    #     print('d_BetaCoef '+str(itr)+str(d_BetaCoef))

    return BetaCoef1

if __name__ == '__main__': # メイン処理

    # cl, cd補間
    alpha = [-32,-30,-28,-26,-24,-22,-20,-18,-16,-14,-12,-10,-8,-6,-4,-2,0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32]

    cl_m3 = [-0.9675, -1.0, -0.996, -0.992, -0.988, -0.984, -0.98,  -0.976, -0.972, -1.07,-0.724,-0.37,-0.19,-0.39,-0.45,-0.19,
              0.03,0.243,0.46,0.67,0.89,1.1,1.25,1.1,0.98,0.9828,0.9856,0.9884,0.9912,0.994,0.997,1,0.9675]
    cl_m4 = [-0.9675,-1,-0.955,-0.96,-0.962,-0.964,-0.966,-0.968,-0.97,-0.82,-0.535,-0.24,-0.3,-0.45,-0.42,-0.185,
              0.05,0.28,0.51,0.75,0.98,1.17,1.13,1.03,0.96,0.9657,0.9714,0.9771,0.9828,0.9885,0.9942,1,0.9675]
    cl_m5 = [-0.9675,-1,-0.94,-0.93,-0.92,-0.925,-0.93,-0.935,-0.94,-0.8,-0.525,-0.4,-0.3,-0.32,-0.44,-0.195,0.05,
              0.295,0.53,0.78,0.96,1.01,0.96,1.08,1.06,1.07,1.06,1.05,1.035,1.02,1.01,1,0.9675]
    cl_m6 = [-0.9675,-1,-0.946,-0.942,-0.938,-0.934,-0.93,-0.926,-0.922,-0.805,-0.66,-0.6,-0.55,-0.52,-0.47,-0.195,0.075,
              0.34,0.613,0.84,0.915,0.947,1,1.054,1.08,1.063,1.053,1.042,1.031,1.02,1.01,1,0.9675]

    cd_m3 = [0.6975,0.63,0.562,0.488,0.417,0.34,0.267,0.195,0.12,0.045,0.018,0.012,0.0082,0.0079,0.0075,0.0075,0.0075,
              0.008,0.0085,0.009,0.011,0.017,0.026,0.145,0.23,0.293,0.345,0.4,0.455,0.507,0.56,0.63,0.6975 ]
    cd_m4 = [0.6975,0.63,0.57,0.51,0.448,0.39,0.33,0.265,0.208,0.161,0.022,0.013,0.009,0.0085,0.008,0.008,0.008,
              0.0082,0.0085,0.011,0.014,0.02,0.098,0.169,0.23,0.293,0.345,0.4,0.455,0.507,0.56,0.63,0.6975 ]
    cd_m5 = [0.6975,0.63,0.564,0.51,0.465,0.408,0.353,0.296,0.24,0.183,0.12,0.067,0.021,0.01,0.008,0.0075,0.0075,
              0.0075,0.008,0.011,0.026,0.08,0.153,0.212,0.262,0.315,0.365,0.416,0.469,0.52,0.569,0.63,0.6975 ]
    cd_m6 = [0.6975,0.63,0.578,0.525,0.469,0.415,0.361,0.323,0.285,0.246,0.191,0.128,0.07,0.026,0.0125,0.0085,0.008,
              0.0085,0.011,0.028,0.073,0.122,0.179,0.231,0.283,0.328,0.358,0.412,0.467,0.521,0.576,0.63,0.6975 ]

    fitted_curve_cl = interpolate.interp1d(alpha, cl_m4)
    fitted_curve_cd = interpolate.interp1d(alpha, cd_m4)

    alpha_high = [ -50,  -48,  -46,  -44,  -42,  -40,  -38,  -36,  -34,  -32,  -30,  -28,  -26,  -24,  -22,  -20,  -18,  -16,  -14,  -12,
                   -10,   -8,   -6,   -4,   -2,    0,    2,    4,    6,    8,   10,   12,   14,   16,   18,   20,   22,   24,   26,   28,
                    30,   32,   34,   36,   38,   40,   42,   44,   46,   48,   50]

    cl_HighAngle = [-0.675, -0.7075, -0.74, -0.7725, -0.805, -0.8375, -0.87, -0.9025, -0.935, -0.9675, -1.0, -0.996, -0.9
                     -0.998, -0.984, -0.98, -0.976, -0.972, -1.07, -0.724, -0.37, -0.19, -0.39, -0.45, -0.19, 0.03,
                     0.243, 0.46, 0.67, 0.89, 1.1, 1.25, 1.1, 0.98, 0.9828, 0.9856, 0.9884, 0.9912, 0.994,
                     0.997, 1.0, 0.9675, 0.935, 0.9025, 0.87, 0.8375, 0.805, 0.7725, 0.74, 0.7075, 0.675]
    cd_HighAngle = [1.335, 1.2625, 1.19, 1.1175, 1.045, 0.9725, 0.9, 0.8325, 0.765, 0.6975, 0.63, 0.562, 0.488,
                    0.417, 0.34, 0.267, 0.195, 0.12, 0.045, 0.018, 0.012, 0.008, 0.00775, 0.0075, 0.0075, 0.0075,
                    0.008, 0.085, .009, .011, 0.017, 0.025, 0.145, 0.23, 0.293, 0.345, 0.4, 0.455, 0.507,
                    0.56, 0.63, 0.6975, 0.765, 0.8325, 0.9, 0.9725, 1.045, 1.1175, 1.19, 1.2625, 1.335]

    fitted_curve_cl_HighAngle = interpolate.interp1d(alpha_high, cl_HighAngle)
    fitted_curve_cd_HighAngle = interpolate.interp1d(alpha_high, cd_HighAngle)

    trim = 1 # =0(input A1s, B1s), =1(calculate A1s, B1s to make a1=b1=0)
    aero = 0 # =0(linear), =1(table)

    omg = 27 #rad/s                                                                                       表3
    r_rotor = 8.178 #m                                                                                    表3
    r_cg = 5.32 #m                                                                                        表3
    e = 0.381 #m                                                                                          表3
    r_a = 4.7 #m                                                                                          表3/第1報(41)式
    S = 3.36 #m^2                                                                                         表3
    rho = 1.225 #kg/m^3                                                                                   表3

    alh = -5.2 * np.pi / 180 #rad                                                                         表2(3行目)
    MT = 0.65                                                                                             表1
    mu = 0.25                                                                                             表2
    Vc = mu * 340 * MT #m/s                                                                               (5)式
    Vs = Vc * np.tan(alh)

    th0 = 8 * np.pi / 180 #rad                                                                            表2(3行目)

    #A1si,B1si初期値
```

```
    A1si = -3.4 * np.pi / 180 #rad                                                                  表2(3行目)
    B1si =  4.0 * np.pi / 180 #rad                                                                  表2(3行目)

    # モデルを定義
    M, D, K = model(omg, r_cg, e)

    # 解析時間
    count = 60  #1周期ごとに計算するデータの数
    dt = 2 * np.pi / omg / count                                                                    第1報(34)式
    t_max = 50
    t = np.arange(0.0, t_max, dt)
    ts = t.reshape(int(t_max / dt + 1), 1)

    x1 = np.cos(omg * ts)
    x2 = np.sin(omg * ts)
    ones = np.ones(len(x1))
    ones = ones.reshape(len(x1), 1)
    X = np.concatenate([ones, x1, x2], 1)

    # 初期条件
    x0 = [0.001, 0.001]
    v0 = [  0.0,   0.0]

    date = datetime.datetime.now().strftime('%m%d')

    filename = date+'_alh='+str(round(alh*180/np.pi, 1))+'_th0='+str(round(th0*180/np.pi, 1))

    d = 0.1 * np.pi / 180 #rad

    j = 0
    itr = 0

    print('初期値:A1s='+str(round(A1si * 180 / np.pi, 2))+' B1s='+str(round(B1si * 180 / np.pi, 2)))

    # a1_i, b1_i
    A1s = A1si
    B1s = B1si
    var = np.concatenate([x0, v0])
    var = odeint(f, var, t, args=(M, D, K, omg, r_a, e, Vc, Vs, j, th0, A1s, B1s, S, rho, aero))
    BetaCoef = Beta_Coef()
    a1_i = BetaCoef[0,1]
    b1_i = BetaCoef[0,2]
    abi = np.array([a1_i, b1_i])

    while np.any(abs(abi) > 0.0005) and trim == 1 :

        # a1_Bj, b1_Bj
        A1s = A1si
        B1s = B1si + d
        var = np.concatenate([x0, v0])
        var = odeint(f, var, t, args=(M, D, K, omg, r_a, e, Vc, Vs, j, th0, A1s, B1s, S, rho, aero))
        BetaCoef = Beta_Coef()
        a1_Bj = BetaCoef[0,1]
        b1_Bj = BetaCoef[0,2]
        time.sleep(0.1)

        # a1_Aj, b1_Aj
        A1s = A1si + d
        B1s = B1si
        var = np.concatenate([x0, v0])
        var = odeint(f, var, t, args=(M, D, K, omg, r_a, e, Vc, Vs, j, th0, A1s, B1s, S, rho, aero))
        BetaCoef = Beta_Coef()
        a1_Aj = BetaCoef[0,1]
        b1_Aj = BetaCoef[0,2]
        time.sleep(0.1)

        # a1_j, b1_j
        A1s = A1si + d
        B1s = B1si + d
        var = np.concatenate([x0, v0])
        var = odeint(f, var, t, args=(M, D, K, omg, r_a, e, Vc, Vs, j, th0, A1s, B1s, S, rho, aero))
        BetaCoef = Beta_Coef()
        a1_j = BetaCoef[0,1]
        b1_j = BetaCoef[0,2]
        time.sleep(0.1)

        P = np.array([[(a1_Bj - a1_i) / d, (a1_Aj - a1_i) / d],                                     (15)式
                      [(b1_Bj - b1_i) / d, (b1_Aj - b1_i) / d]])                                    (15)式

        P_inv = np.linalg.inv(P)                                                                    (14)式
        dABi = - P_inv @ abi                                                                        (14)式

        # 更新
        B1si = B1si + 0.3 * dABi[0]                                                                 (13)式
        A1si = A1si + 0.3 * dABi[1]                                                                 (13)式

        # a1_i, b1_i
        A1s = A1si
        B1s = B1si
        var = np.concatenate([x0, v0])
        var = odeint(f, var, t, args=(M, D, K, omg, r_a, e, Vc, Vs, j, th0, A1s, B1s, S, rho, aero))
        BetaCoef = Beta_Coef()
        a1_i = BetaCoef[0,1]
        b1_i = BetaCoef[0,2]
        abi = np.array([a1_i, b1_i])
        time.sleep(0.1)

        a1b1 = np.array([[a1_j, b1_j]])
        itr = itr +1
        print('while itr ='+str(itr),' A1s='+str(round(A1si*180/np.pi,3)),
              ' B1s='+str(round(B1si*180/np.pi,3)), ' a1,b1: '+str(abi*180/np.pi))
```

```
344
345     A1sB1s = np.array([[A1si, B1si]])
346     a1b1 = np.array([[a1_i, b1_i]])
347     np.savetxt(filename+'_A1sB1s.txt',A1sB1s)
348     np.savetxt(filename+'_a1b1.txt', a1b1)
349     time.sleep(0.1)
350
351     for j in [0, 1, 2, 3]:
352         A1s = A1si
353         B1s = B1si
354
355         var = np.concatenate([x0, v0])
356
357         # 微分方程式の近似解法(過渡応答)
358         var = odeint(f, var, t, args=(M, D, K, omg, r_a, e, Vc, Vs, j, th0, A1s, B1s, S, rho, aero))
359
360         Fw_list = []
361         Fb_list = []
362         Fhb_list = []
363         Fhr_list = []
364         Fhf_list = []
365
366         # varを元にvar2を計算
367         for i in np.arange(0, len(var)):
368             Fw, Fb, Fhb, Fhr, Fhf = f2(var[i], t[i], omg, r_a, e, Vc, Vs, j, th0, A1s, B1s, S, rho, aero)
369             Fw_list.append(Fw)
370             Fb_list.append(Fb)
371             Fhb_list.append(Fhb)
372             Fhr_list.append(Fhr)
373             Fhf_list.append(Fhf)
374
375         var_j = np.array(var)
376         Fw_j = np.array(Fw_list)
377         Fb_j = np.array(Fb_list)
378         Fhb_j = np.array(Fhb_list)
379         Fhr_j = np.array(Fhr_list)
380         Fhf_j = np.array(Fhf_list)
381
382         Q = 0.75 * r_rotor* Fhr_j[:, 0]      #   Q = 0.5 * r_rotor**2 * Fhr_j[:, 0]       2025/06/16 nakayama             表3/第1報(52)式
383         Q_j = np.array([Q])
384         Q_j = Q_j.reshape(int(len(Fhr_list)), 1)
385
386         if j == 0: #j=1のときj1のデータを保存
387             BetaCoef_j1 = Beta_Coef()
388             np.savetxt(filename+'_BetaCoef.txt', BetaCoef_j1)
389
390             var_all = var_j
391             Fw_all = Fw_j
392             Fb_all = Fb_j
393             Fhb_all = Fhb_j
394             Fhr_all = Fhr_j
395             Fhf_all = Fhf_j
396             Qj_all = Q_j
397             Q_all = Q_j
398
399         else:
400             var_all = np.hstack((var_all, var_j))
401             Fw_all = np.hstack((Fw_all, Fw_j))
402             Fb_all = np.hstack((Fb_all, Fb_j))
403             Fhb_all = np.hstack((Fhb_all, Fhb_j))
404             Fhr_all = np.hstack((Fhr_all, Fhr_j))
405             Fhf_all = np.hstack((Fhf_all, Fhf_j))
406             Qj_all = np.hstack((Qj_all, Q_j))
407             Q_all += Q_j   # Q_all =+ Q_j   2025/06/16 nakayama
408
409     # ブレード1枚ごとの計算結果
410     # np.savetxt(date+'_angle_j.txt', np.concatenate([ts, var_all], 1))   # [t, j1ξ, j1β, j1ξ', j1β', j2ξ, j2β, j2ξ
411     # np.savetxt(date+'_Fw_j.txt', np.concatenate([ts, Fw_all], 1))       # [t, j1Fwx, j1Fwy, j1Fwz, j2Fwx, j2Fwy, j2Fwz,
412     # np.savetxt(date+'_Fb_j.txt', np.concatenate([ts, Fb_all], 1))       # [t, j1Fbx, j1Fby, j1Fbz, j2Fbx, j2Fby, j2Fbz,
413     # np.savetxt(date+'_Fhb_j.txt', np.concatenate([ts, Fhb_all], 1))     # [t, j1Fhbx, j1Fhby, j1Fhbz, j2Fhbx, j2Fhby, j
414     # np.savetxt(date+'_Fhr_j.txt', np.concatenate([ts, Fhr_all], 1))     # [t, j1Fhrx, j1Fhry, j1Fhrz, j2Fhrx, j2Fhry, j
415     # np.savetxt(date+'_Fhf_j.txt', np.concatenate([ts, Fhf_all], 1))     # [t, j1Fhfx, j1Fhfy, j1Fhfz, j2Fhfx, j2Fhfy, j
416     # np.savetxt(date+'_Q_j.txt', np.concatenate([ts, Qj_all], 1))        # [t, j1Q, j2Q, j3Q, j4Q]
417
418     # 4枚のブレードのFx, Fy, Fzをそれぞれ合計してローターの空気力を求める
419     Fx_h = Fhf_all[:, 0] + Fhf_all[:, 3] + Fhf_all[:, 6] + Fhf_all[:, 9]
420     Fy_h = Fhf_all[:, 1] + Fhf_all[:, 4] + Fhf_all[:, 7] + Fhf_all[:, 10]
421     Fz_h = Fhf_all[:, 2] + Fhf_all[:, 5] + Fhf_all[:, 8] + Fhf_all[:, 11]
422
423     Fx = Fx_h*np.cos(alh)+Fz_h*np.sin(alh)
424     Fy = Fy_h
425     Fz = Fx_h*-np.sin(alh)+Fz_h*np.cos(alh)
426     Fx = Fx.reshape(len(Fhf_all), 1)
427     Fy = Fy.reshape(len(Fhf_all), 1)
428     Fz = Fz.reshape(len(Fhf_all), 1)
429     F_all = np.concatenate([Fx, Fy, Fz], 1)
430
431     angle_j1 = np.array(var_all[:, 0:2])    #j1の[ξ, β] [0:2]
432     angle_j2 = np.array(var_all[:, 4:6])    #j2の[ξ, β] [4:6]
433     angle_j3 = np.array(var_all[:, 8:10])   #j3の[ξ, β] [8:10]
434     angle_j4 = np.array(var_all[:, 12:14])  #j4の[ξ, β] [12:14]
435     angle_all = np.hstack((angle_j1, angle_j2, angle_j3, angle_j4))
436
437     np.savetxt(filename+'_angle.txt', np.concatenate([angle_all], 1))   # [j1ξ, j1β, j2ξ, j2β, j3ξ, j3β, j4ξ, j4β
438     np.savetxt(filename+'_Fw.txt', np.concatenate([Fw_all], 1))     # [Fwx, Fwy, Fwz]       相対風座標系
439     np.savetxt(filename+'_Fb.txt', np.concatenate([Fb_all], 1))     # [Fbx, Fby, Fbz]       ブレード固定座標系
440     np.savetxt(filename+'_Fhb.txt', np.concatenate([Fhb_all], 1))   # [Fhbx, Fhby, Fhbz]    ハブブレード中間座標系
441     np.savetxt(filename+'_Fhr.txt', np.concatenate([Fhr_all], 1))   # [Fhrx, Fhry, Fhrz]    ハブ回転座標系
442     np.savetxt(filename+'_Fhf.txt', np.concatenate([Fhf_all], 1))   # [Fhfx, Fhfy, Fhfz]    ハブ非回転座標系
443     np.savetxt(filename+'_F.txt', np.concatenate([F_all], 1))       # [Fx, Fy, Fz]          慣性系
444     np.savetxt(filename+'_Q.txt', np.concatenate([Q_all], 1))       # [Q]                   トルク
```