

# Algorithmic Outline of Unsteady Aerodynamics (AERODYN) Modules

Project WE-201103

## APPENDICES

September 2, 2011

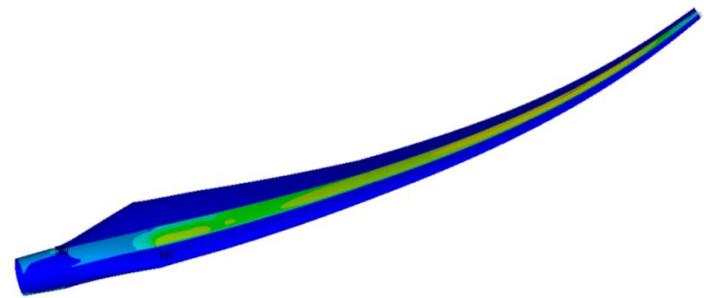
PREPARED FOR:

Alliance For Sustainable Energy, LLC  
Management and Operating Contractor for  
the National Renewable Energy Laboratory  
("NREL")

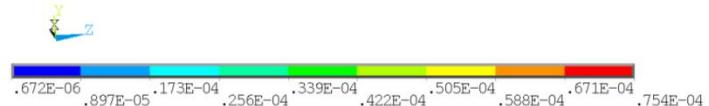


Subcontract No. AFT-1-11326-01  
Under  
Prime Contract No. DE-AC36-08GO28308

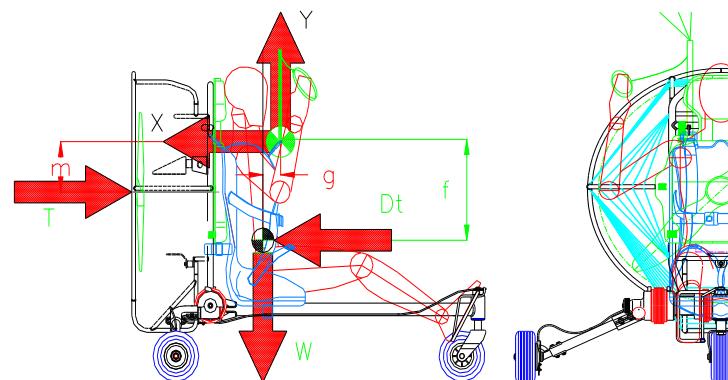
Contract Technical Monitor: Patrick Moriarty  
Contract Administrator: Heidi Oliver



PREPARED BY:  
Rick Damiani, Ph.D., P.E.  
[rdamiani@RRDEngineering.com](mailto:rdamiani@RRDEngineering.com)  
Tel: +1-970-581-8091



**RRD**  
ENGINEERING  
670 Cody St.  
Lakewood, Colorado 80215, USA



<b>APPENDIX A. HIGH LEVEL FLOWCHARTS.....</b>	<b>A-1</b>
A1. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE AERO>DYN .....	A-2
A2. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE AERO>SUBS .....	A-5
A3. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE AERO>GENSUBS .....	A-22
A4. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE INFLOW>WIND.....	A-23
A5. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE HH>WIND.....	A-30
A6. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE FFW>WIND.....	A-32
A7. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE FD>WIND .....	A-35
A8. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE CT>WIND .....	A-39
A9. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE FD>WIND .....	A-42
<b>APPENDIX B. AERODYN MODULE .....</b>	<b>B-1</b>
B1. AD_INIT(ADOPTIONS, TURBINECOMPONENTS, ERRSTAT).....	B-3
B2. AD_GETCONSTANT (VARNAMEERRSTAT) .....	B-7
B3. AD_GETCURRENTVALUE (VARNAME, ERRSTAT, IBLADE, IELEMENT).....	B-8
B4. AD_GETUNDISTURBEDWIND (TIME, INPUTPOSITION,ERRSTAT) .....	B-9
B5. AD_TERMINATE (ERRSTAT) .....	B-10
B6. AD_CALCULATELOADS(CURRENTTIME, INPUTMARKERS, TURBINECOMPONENTS, CURRENTADOPTIONS, ERRSTAT) .....	B-11
<b>APPENDIX C. AEROSUBS MODULE.....</b>	<b>C-1</b>
C1. READFL() .....	C-4
C2. AD_GETINPUT (UNIN, AEROINFILE, WINDFILENAME, TITLE, ERRSTAT).....	C-7
C3. READTWR (UNIN, FILENAME, ERRSTAT) .....	C-11
C4. CHECKRCOMP(ADFILE, HUBRADIUS, TIPRADIUS, ERRSTAT) .....	C-12
C5. ADOUT (TITLE, HUBRAD, WINDFILENAME).....	C-14
C6. BDDAT() .....	C-16
C7. BEDINIT(J,IBLADE,ALPHA) .....	C-19
C8. BEDDOES(W2,J,IBLADE,ALPHA,CLA,CDA,CMA) .....	C-24
C9. ATTACH(VREL,J,IBLADE,CNA1,AOL1,AE) .....	C-27
C10. SEPAR(NFT,J,IBLADE,IFOIL,CNA1,AOL1,CNS1,CNSL1).....	C-31
C11. VORTEX(J,IBLADE,AE) .....	C-37
C12. BEDUPDATE .....	C-40
C13. BEDWRT .....	C-41
C14. DISKVEL () .....	C-42
C15. AD_WINDVELOCITYWITHDISTURBANCE(INPUTPOSITION, ERRSTAT).....	C-44
C16. GETTWRINFLUENCE(VX,VY, INPUTPOSITION ) .....	C-47
C17. GETTWRSECTPROP(INPUTPOSITION, VELHOR, TWRELRAD, TWRELCD).....	C-51
C18. GETREYNOLDS (WINDSPED, CHORDLEN) .....	C-53
C19. AERODYN_TERMINATE.....	C-54
C20. ABPRECOR(F, OLDF, DFDT, DT, N, NO) .....	C-56
C21. ELEMFRFC (PSI, RLOCAL, J, IBLADE, VNROTOR2, VT, VNW, VNB, DFN, DFT, PMA, INITIAL).....	C-58
C22. VIND (J, IBLADE, RLOCAL, VNROTOR2, VNW, VNB, VT) .....	C-62
C23. VINDERR (VNW, VX, VID, J, IBLADE) .....	C-66

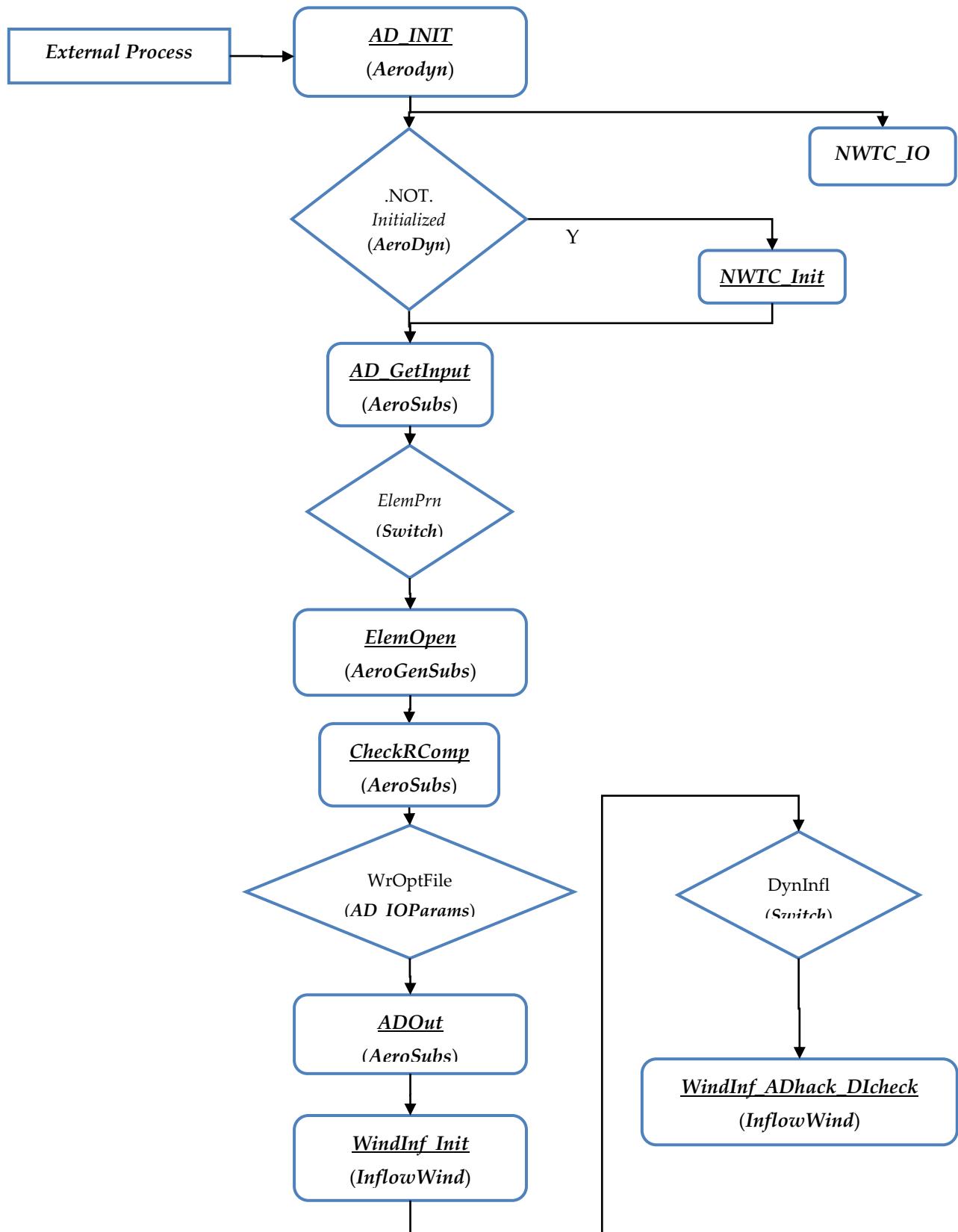
C24. AXIND (VNW, VNB, VNA, VTA, VT, VT2_INV, VNROTOR2, A2, A2P, J, SOLFACT, ALPHA, PHI, CLA, CDA, CMA, RLOCAL) .....	C-67
C25. GETPRANDTLLOSS (LCNST, SPHI, PRLOSS) .....	C-70
C26. GETTIPLOSS (J, SPHI, TIPLOSS, RLOCAL).....	C-71
C27. CLCD (ALPHA, CLA,CDA, CMA, I, ERRSTAT) .....	C-73
C28. VNMOD(J, IBLADE,RLOCAL,PSI) .....	C-76
C29. SAT(X, VAL, SLOPE).....	C-77
C30. INFLOW() .....	C-78
C31. GETRM (RLOCAL, DFN, DFT, PSI,J, IBLADE).....	C-79
C32. INFINIT () .....	C-81
C33. INFDIST () .....	C-86
C34. LMATRIX (X,MATRIXMODE) .....	C-90
C35. GETPHILQ ().....	C-92
C36. WINDAZIMUTHZERO (PSI,WINDPSI) .....	C-94
C37. XPHI(RZERO,MODE) .....	C-95
C38. MATINV (A0, A1, N, NO, INVMODE) .....	C-96
C39. GAUSSJ (A, N) .....	C-98
C40. FGAMMA (R,J,M,N).....	C-101
C41. HFUNC (M,N).....	C-102
C42. IDUBFACT (I).....	C-103
C43. PHIS (RZERO, R, J) .....	C-104
C44. VINDINF (IRADIUS, IBLADE, RLOCAL, VNW, VNB, VT, PSI).....	C-105
C45. DYNDEBUG (RHSCOS, RHSSIN) .....	C-107
C46. INFUPDT ().....	C-109
<b>APPENDIX D. AEROMODS MODULE.....</b>	<b>D-1</b>
<b>APPENDIX E. AEROGENSUBS MODULE .....</b>	<b>E-15</b>
E1. ELEMOPEN(ELEMFILE) .....	E-16
E2. ELEMOUT() .....	E-18
E3. ALLOCARRAYS(ARG) .....	E-19
<b>APPENDIX F. SHAREDTYPES MODULE .....</b>	<b>F-1</b>
<b>APPENDIX G. INFLOWWIND MODULE.....</b>	<b>G-1</b>
G1. WINDINF_INIT(FILEINFO, ERRSTAT) .....	G-3
G2. WINDINF_GETVELOCITY(TIME, INPUTPOSITION, ERRSTAT) .....	G-5
G3. WINDINF_GETMEAN(STARTTIME, ENDTIME, DELT_TIME, INPUTPOSITION, ERRSTAT) .....	G-6
G4. WINDINF_GETSTDDEV(STARTTIME, ENDTIME, DELT_TIME, INPUTPOSITION, ERRSTAT) .....	G-8
G5. WINDINF_GETTI(STARTTIME, ENDTIME, DELT_TIME, INPUTPOSITION, ERRSTAT) .....	G-10
G6. WINDINF_ADHACK_DICHECK (ERRSTAT) .....	G-12
G7. WINDINF_TERMINATE( ERRSTAT) .....	G-13
G8. GETWINDTYPE(FILENAME, ERRSTAT) .....	G-14
G9. WINDINF_ADHACK_DISKVEL(TIME, INPPOSITION, ERRSTAT) .....	G-16
G10. WINDINF_LINEARIZEPERTURBATION(LINPERTURBATIONS, ERRSTAT) .....	G-18

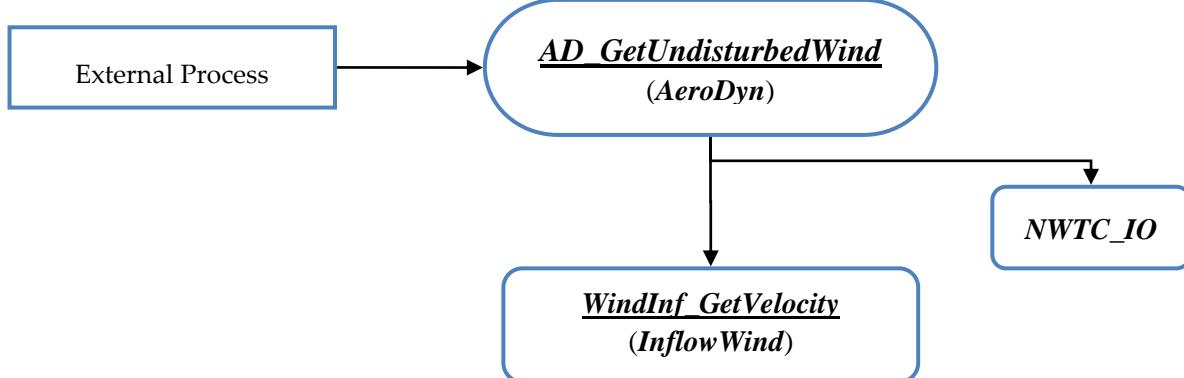
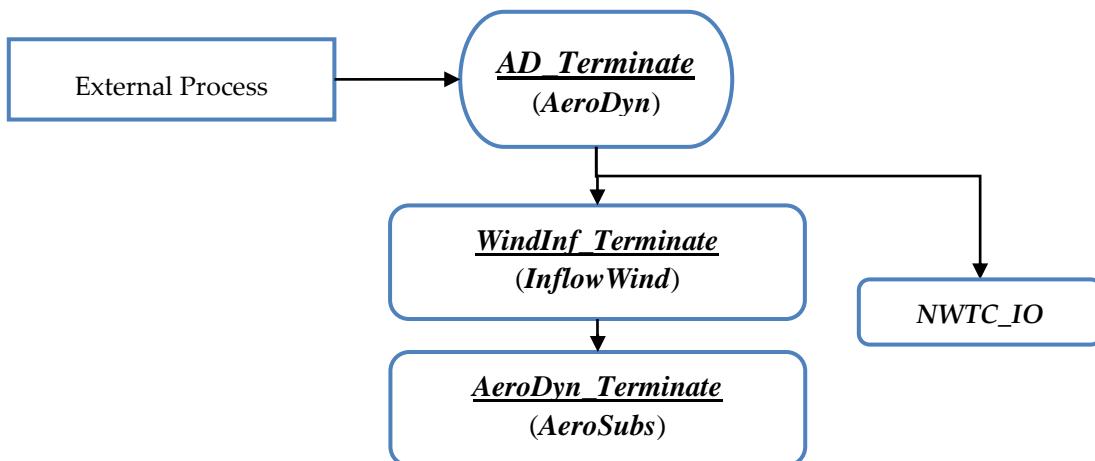
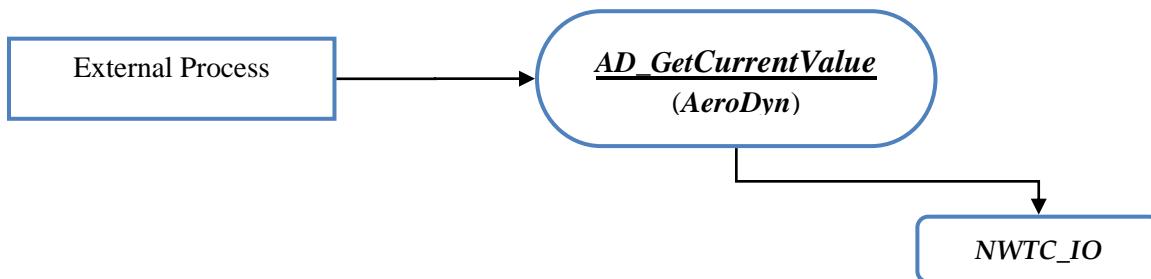
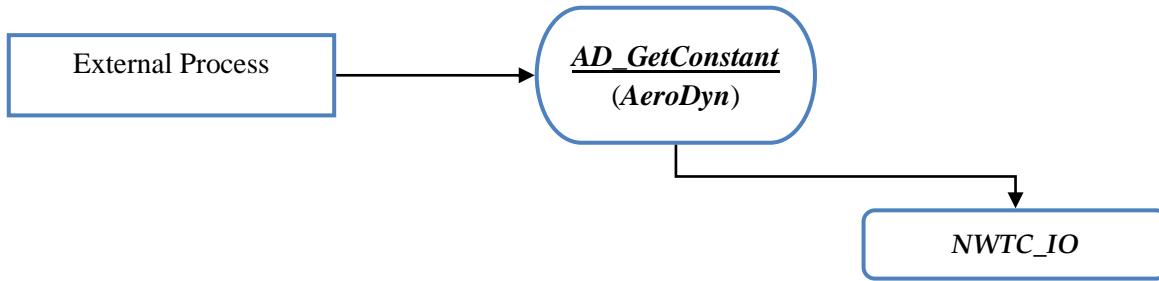
<b>APPENDIX H. SHAREDINFLOWDEFS MODULE .....</b>	<b>H-1</b>
<b>APPENDIX I. HHWIND MODULE.....</b>	<b>I-1</b>
I1. HHINIT (UNWIND, WINDFILE, WINDINFO, ERRSTAT).....	I-3
I2. HH_SETLINEARIZEDDELS (PERTURBATIONS, ERRSTAT).....	I-6
I3. HH_GETWINDSPEED (TIME, INPUTPOSITION, ERRSTAT).....	I-7
I4. HH_GET_ADHACK_WINDSPEED (TIME, INPUTPOSITION, ERRSTAT).....	I-10
I5. HH_TERMINATE (ERRSTAT) .....	I-12
<b>APPENDIX J. FFWIND MODULE.....</b>	<b>J-1</b>
J1. FFINIT (UNWIND, BINFILE, ERRSTAT) .....	J-3
J2. READBLADED_FF_HEADER0 (UNWIND, ERRSTAT) .....	J-6
J3. READBLADED_FF_HEADER1 (UNWIND, TI, ERRSTAT).....	J-8
J4. READ_BLADED_GRIDS (UNWIND, CWISE, TI, ERRSTAT).....	J-10
J5. READ_SUMMARY_FF(UNWIND, FILENAME, CWISE, ZCENTER, TI, ERRSTAT).....	J-12
J6. READ_FF_TOWER(UNWIND, WINDFILE, ERRSTAT) .....	J-15
J7. READ_TURBSIM_FF(UNWIND, WINDFILE, ERRSTAT) .....	J-18
J8. FF_GETWINDSPEED (TIME, INPUTPOSITION, ERRSTAT) .....	J-21
J9. FF_INTERP (TIME, POSITION, ERRSTAT) .....	J-22
J10. FF_GETRVALUE(RVARNAME, ERRSTAT).....	J-27
J11. FF_TERMINATE (ERRSTAT).....	J-28
<b>APPENDIX K. FDWIND MODULE .....</b>	<b>K-1</b>
K1. FDINIT (UNWIND, WINDFILE, REFHT, ERRSTAT) .....	K-6
K2. READFDP(UNWIND, FILENAME, FDTSFILE, ERRSTAT) .....	K-9
K3. READ4DTIMES (UNWIND, FILENAME, ERRSTAT) .....	K-12
K4. READALL4DDATA (UNWIND, ERRSTAT).....	K-13
K5. LOADLES DATA (UNWIND, FILENO, IDX, ERRSTAT) .....	K-14
K6. READ4DDATA(UNWIND, FILENAME, COMP, IDX4, SCALE, OFFSET, ERRSTAT).....	K-15
K7. LOAD4DDATA(INPINDX).....	K-17
K8. FD_GETWINDSPEED (TIME, INPUTPOSITION, ERRSTAT).....	K-18
K9. FD_GETRVALUE(RVARNAME, ERRSTAT) .....	K-23
K10. FD_TERMINATE (ERRSTAT).....	K-24
<b>APPENDIX L. CTWIND MODULE.....</b>	<b>L-1</b>
L1. CTINIT (UNWIND, WINDFILE, BACKGRNDVALUES, ERRSTAT) .....	L-5
L2. READCTP (UNWIND, FILENAME, CTPSCALING, ERRSTAT).....	L-8
L3. READCTTS (UNWIND, FILENAME, CT_SC_EXT, ERRSTAT).....	L-10
L4. READCTSCALES(UNWIND, FILENAME, ERRSTAT).....	L-12
L5. CT_SETREFVAL(HEIGHT, HWIDTH, ERRSTAT) .....	L-13
L6. CT_GETWINDSPEED (TIME, INPUTPOSITION, ERRSTAT).....	L-15
L7. READCTDATA (UNWIND, CTFILENO, ITIME, ERRSTAT) .....	L-19
L8. LOADCTDATA (UNWIND, FILENAME, ITIME, ICOPPM, VEL, ERRSTAT).....	L-20
L9. CT_TERMINATE (ERRSTAT) .....	L-22

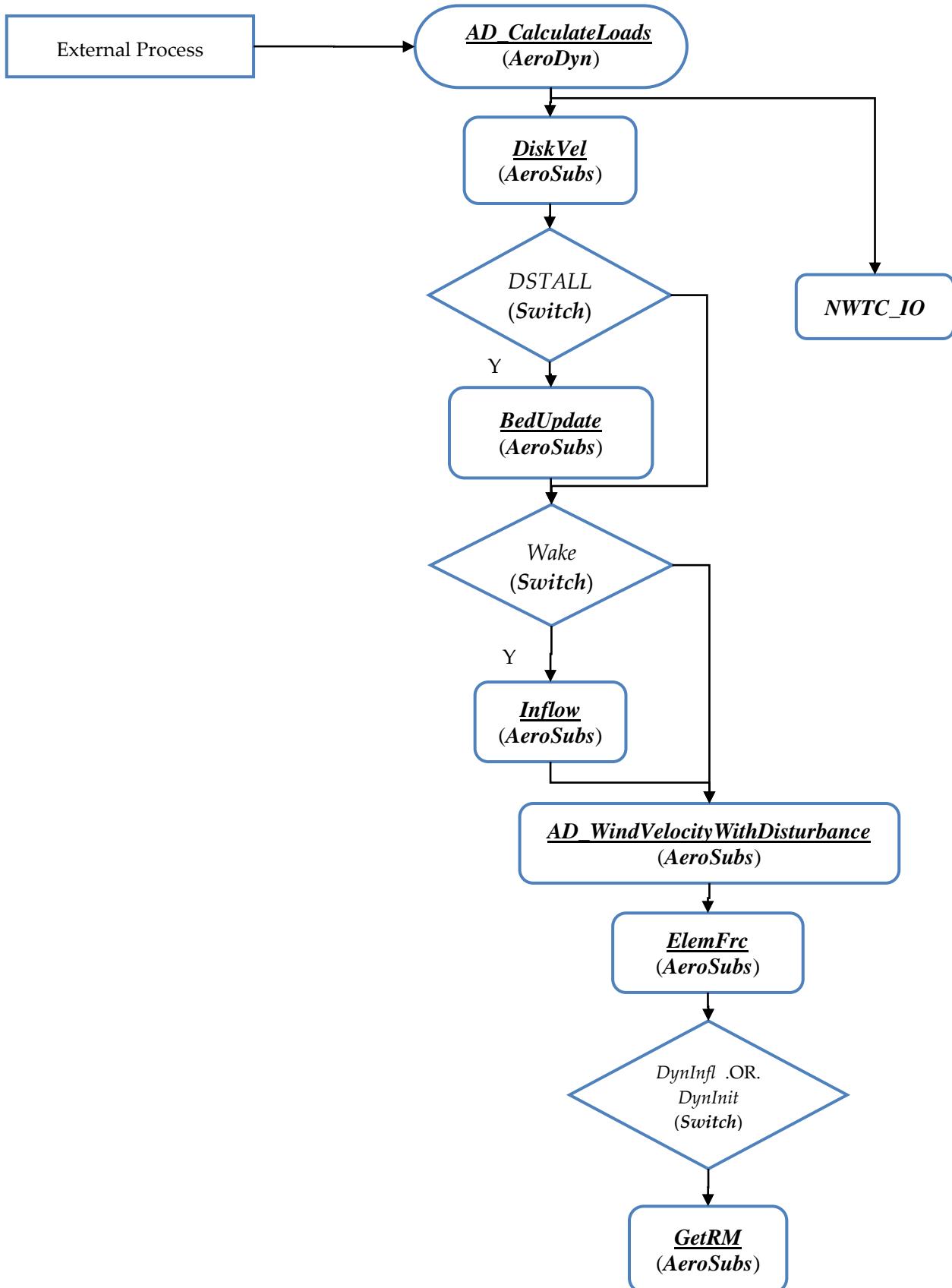
<b>APPENDIX M. USERWIND MODULE .....</b>	<b>M-1</b>
M1. USRWND_INIT (ERRSTAT) .....	M-2
M2. USRWND_GETVALUE(VARNAME, ERRSTAT).....	M-3
M3. USRWND_GETWINDSPEED (TIME, INPUTPOSITION, ERRSTAT).....	M-4
M4. USRWND_TERMINATE (ERRSTAT) .....	M-5
<b>APPENDIX N. DIRECTION COSINES .....</b>	<b>N-1</b>

## APPENDIX A. HIGH LEVEL FLOWCHARTS

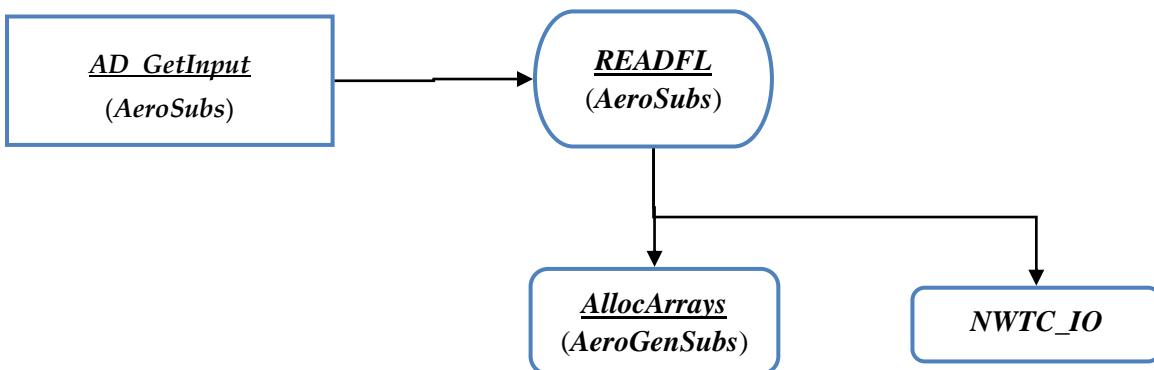
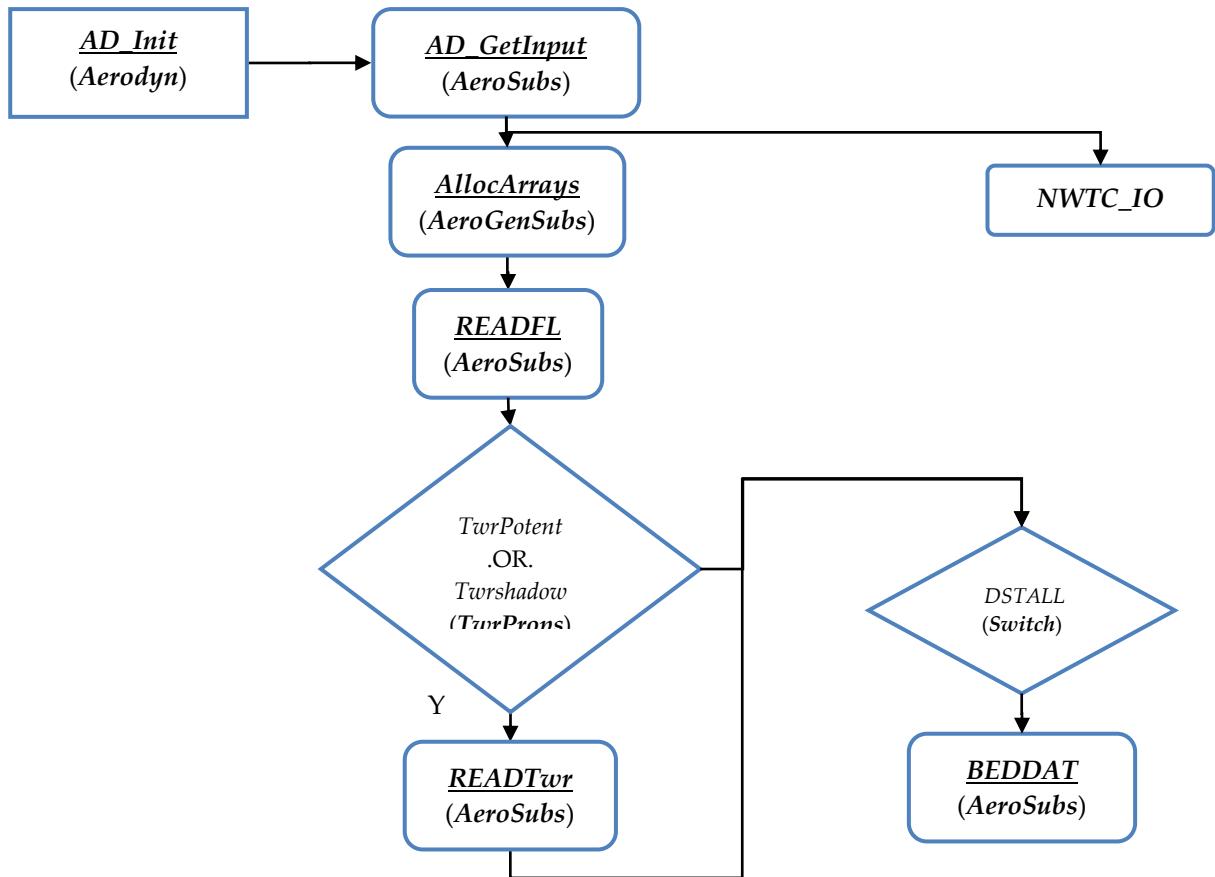
## A1. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE AERODYN

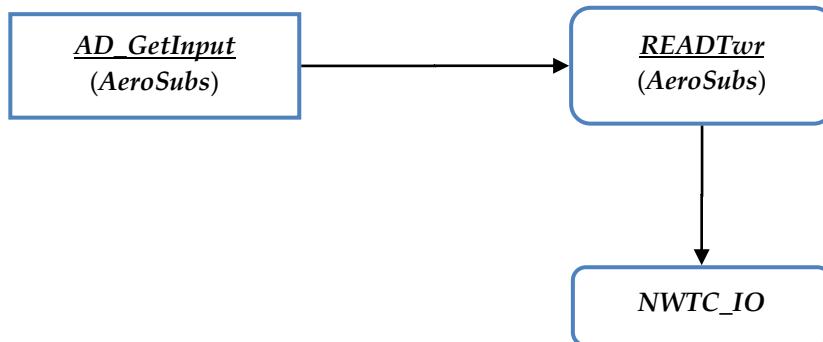
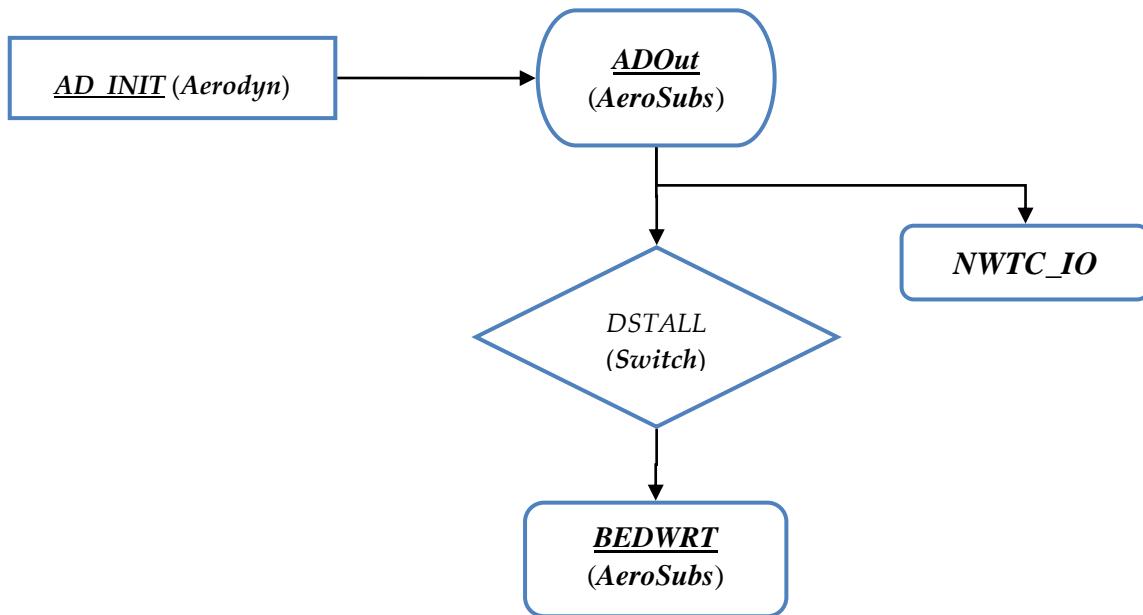
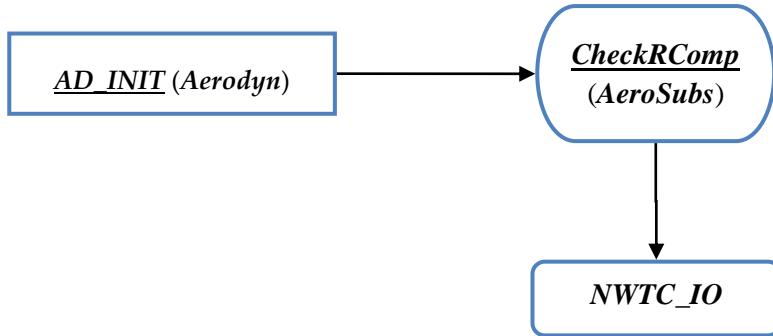


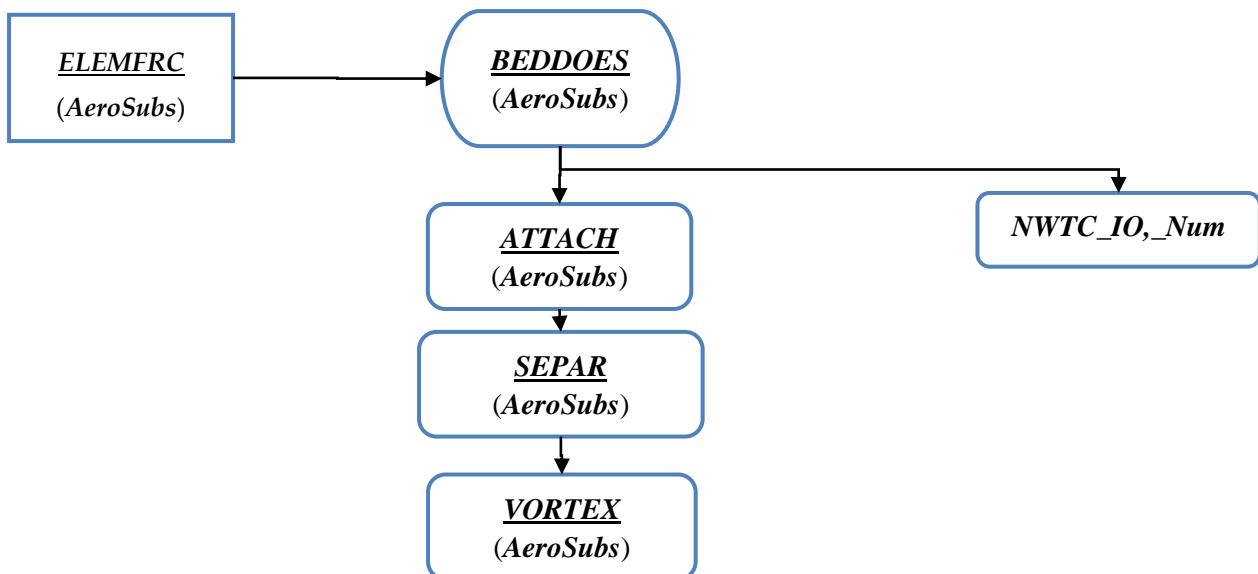
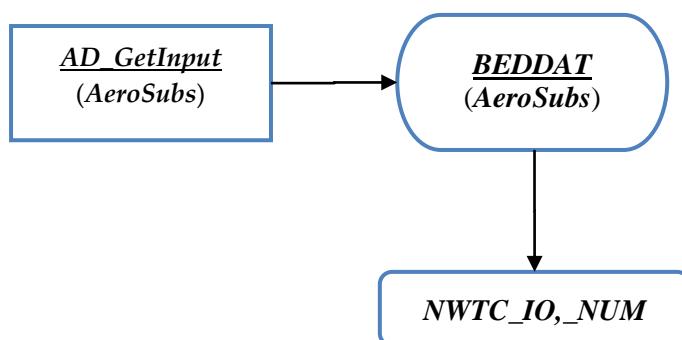
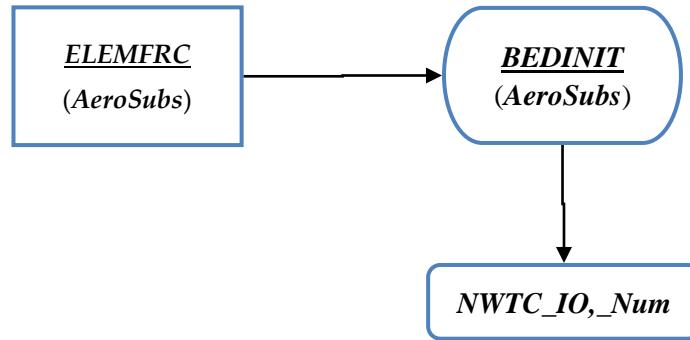


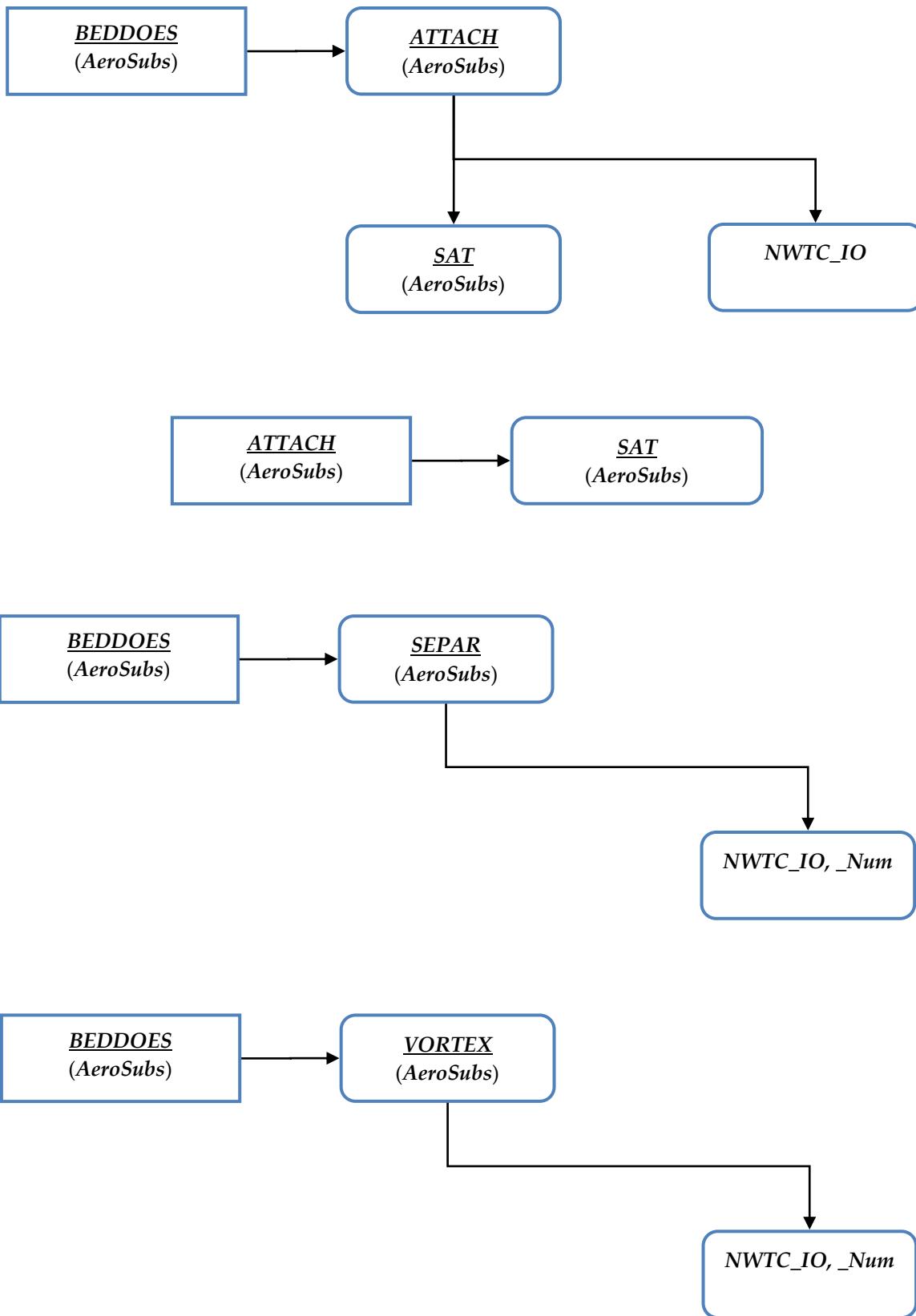


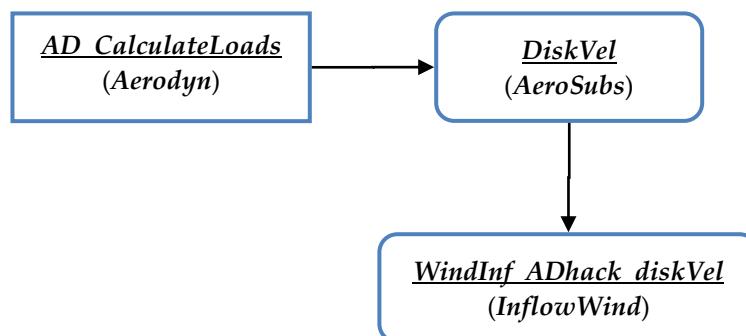
## A2. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE AEROSUBS

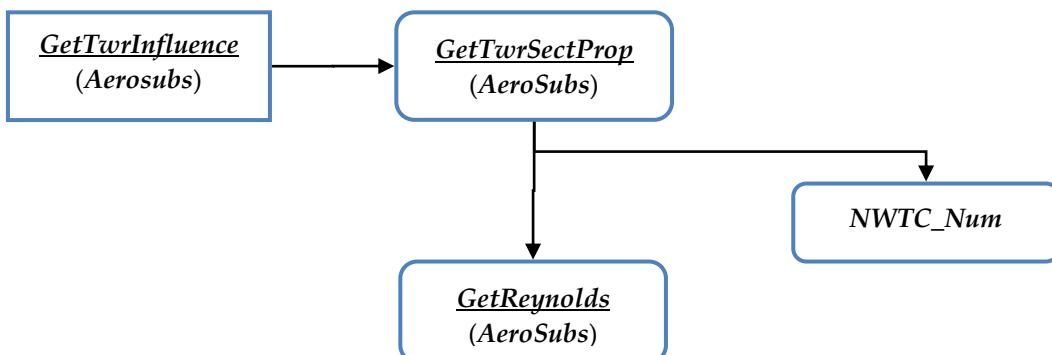
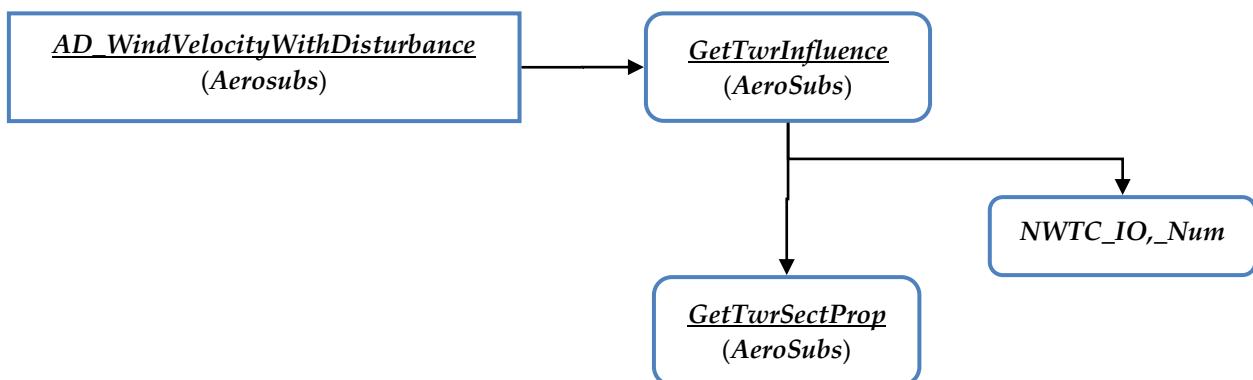
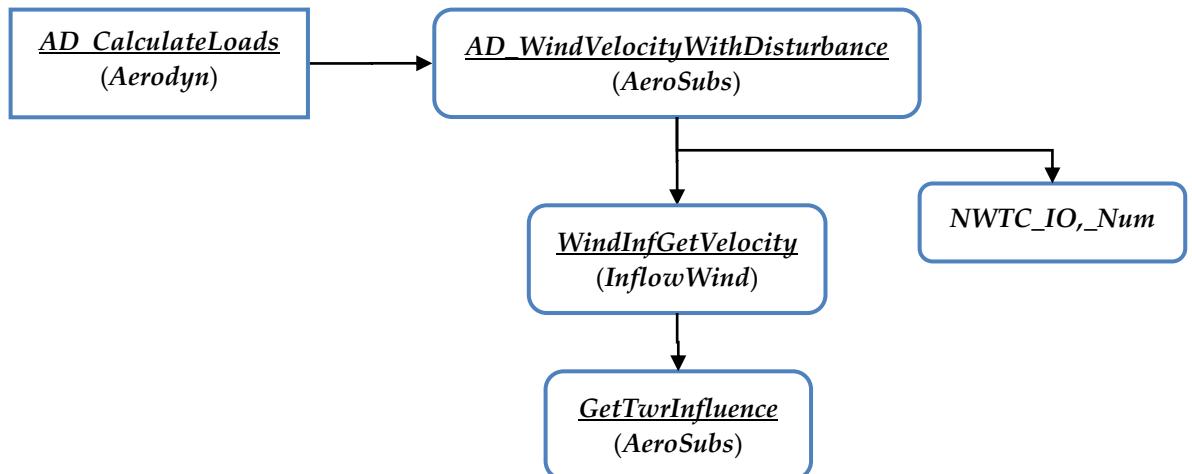


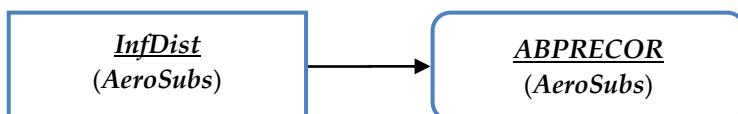
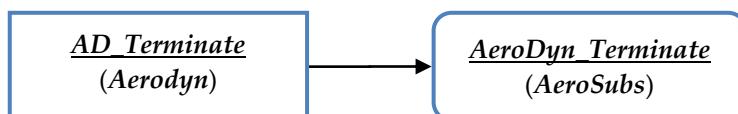
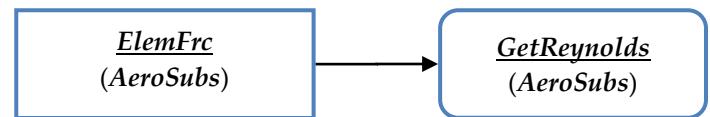


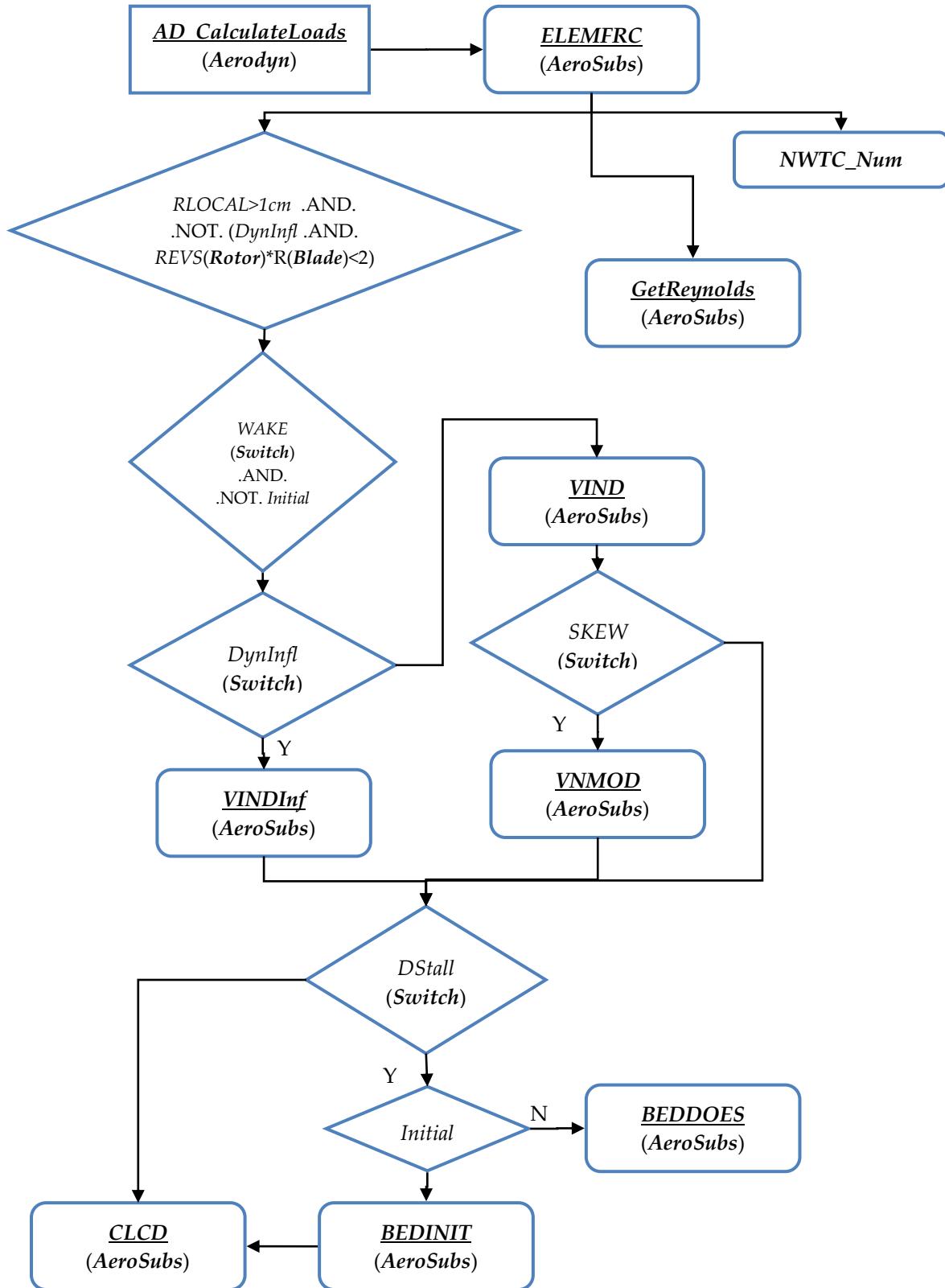


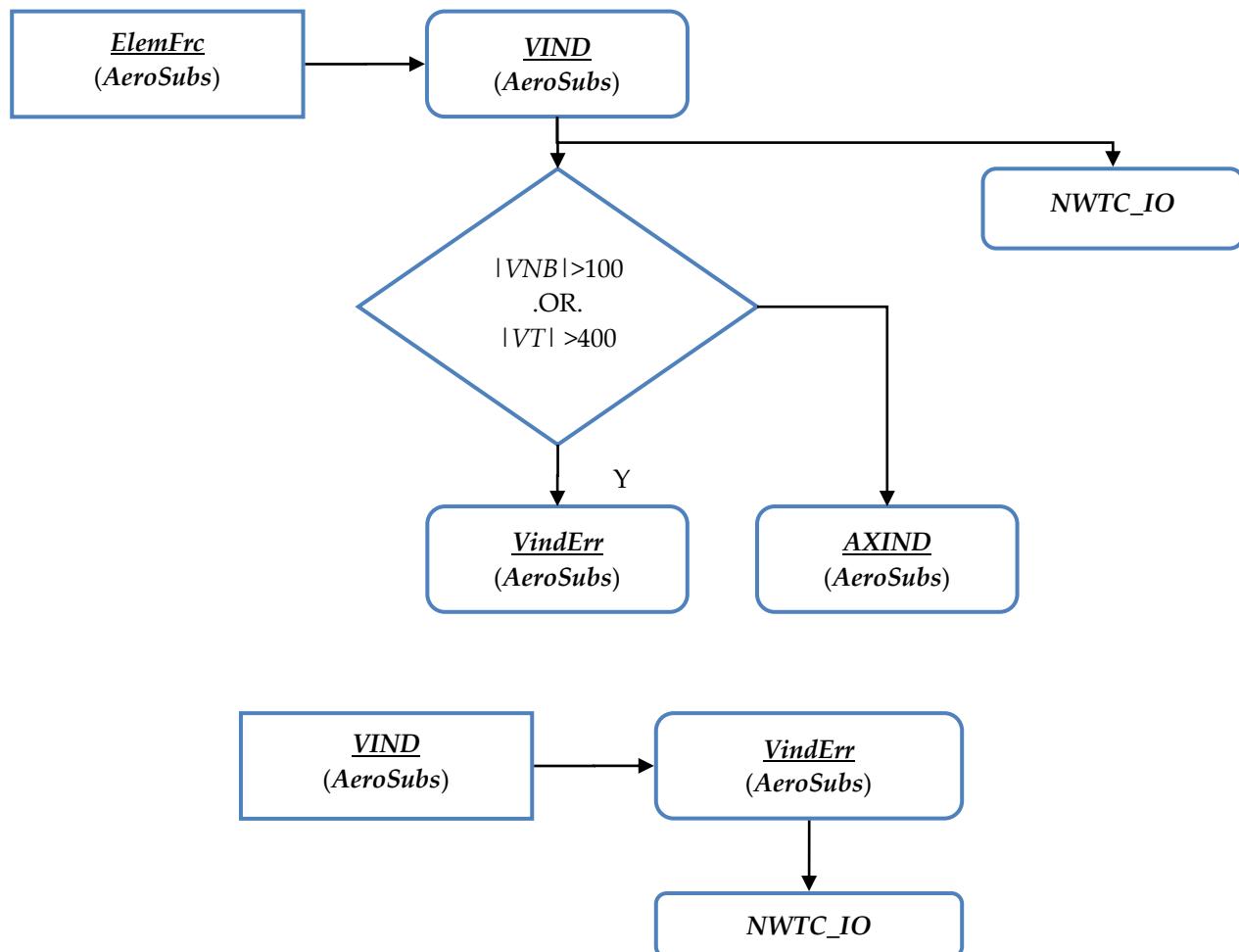


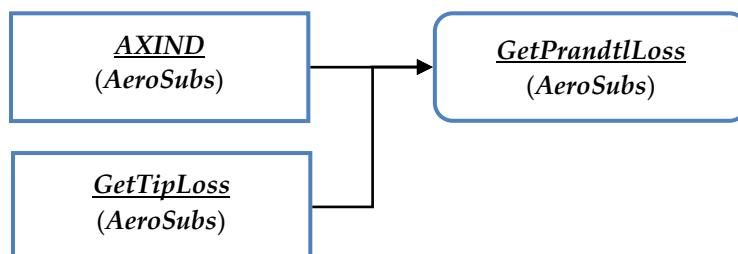
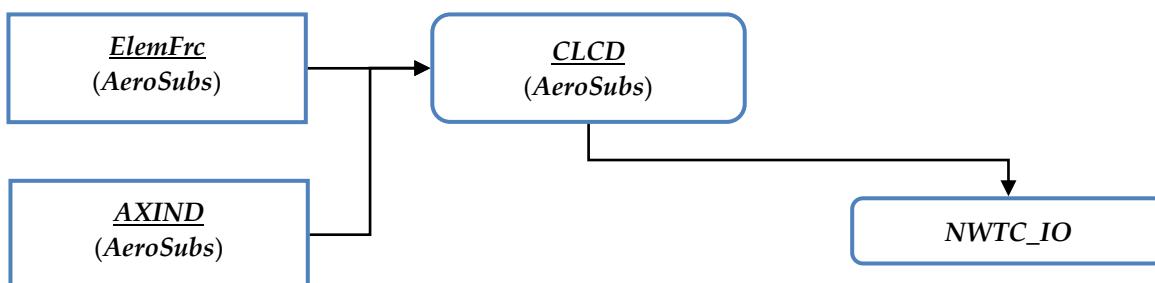
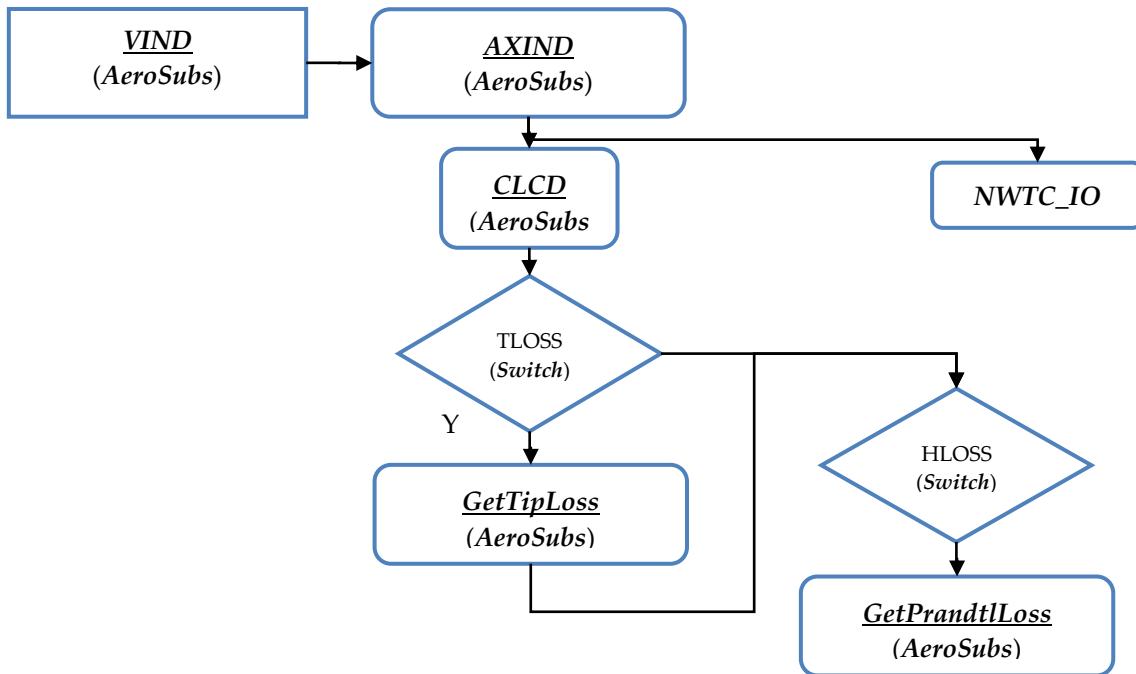


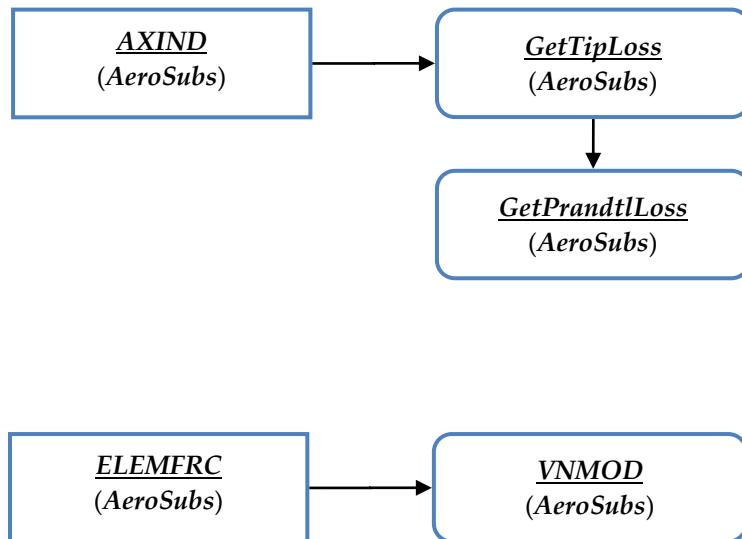


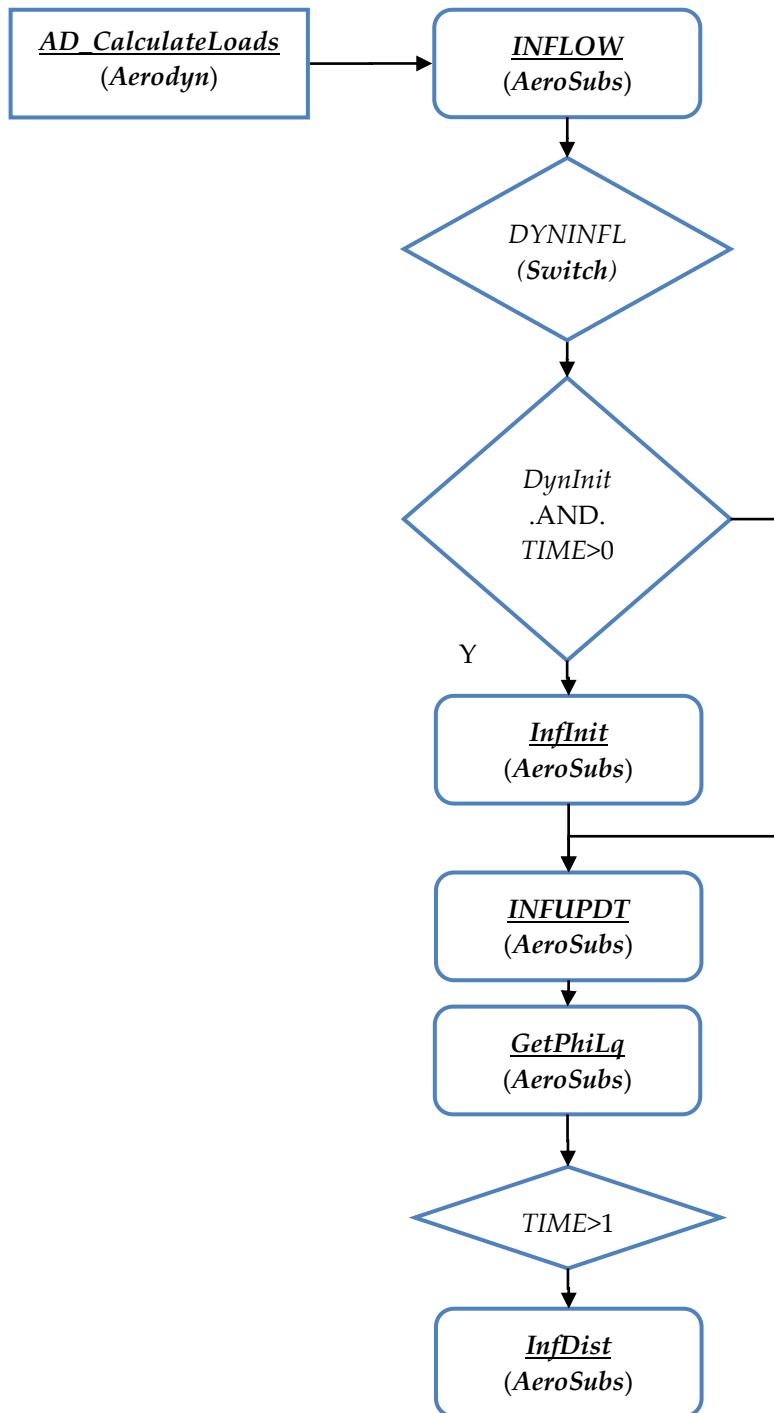


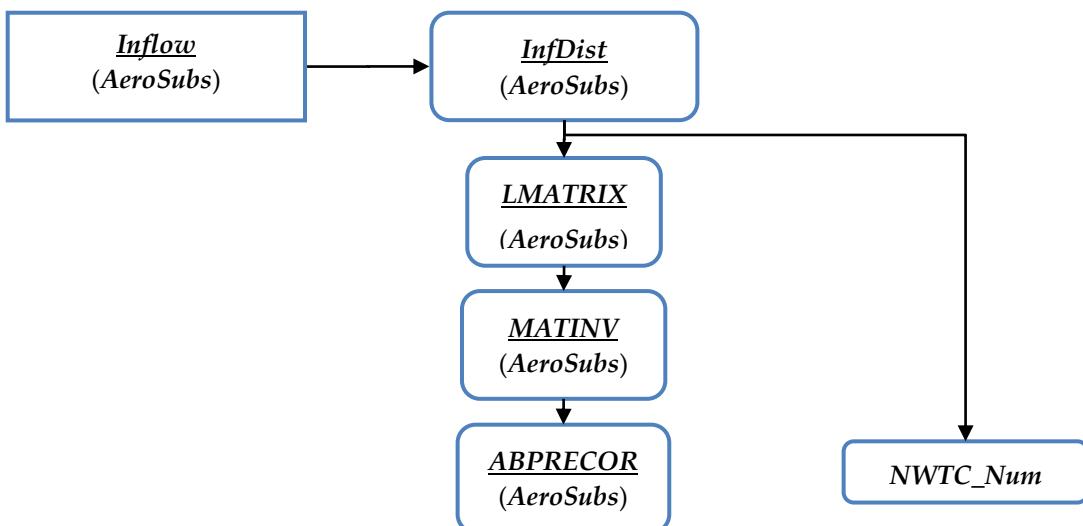
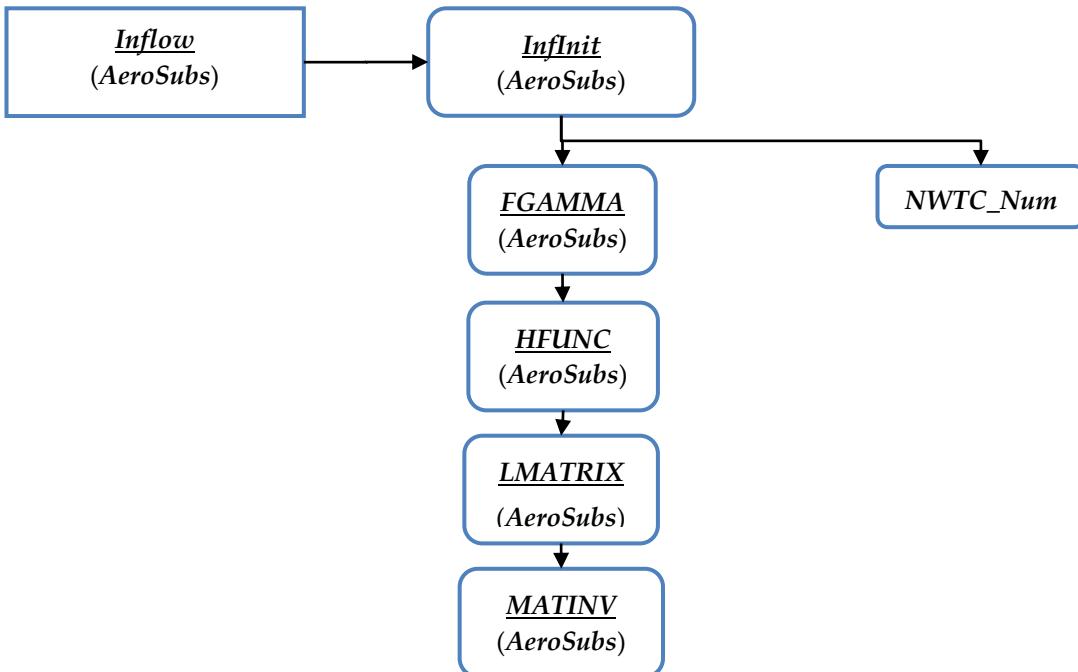


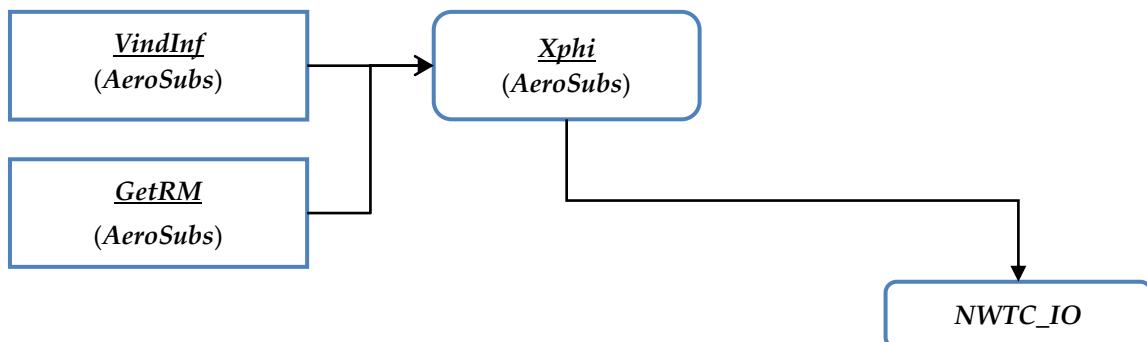
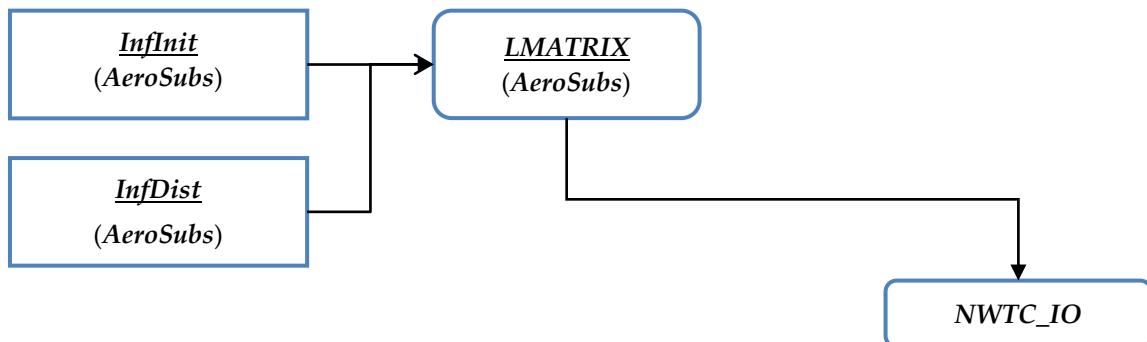
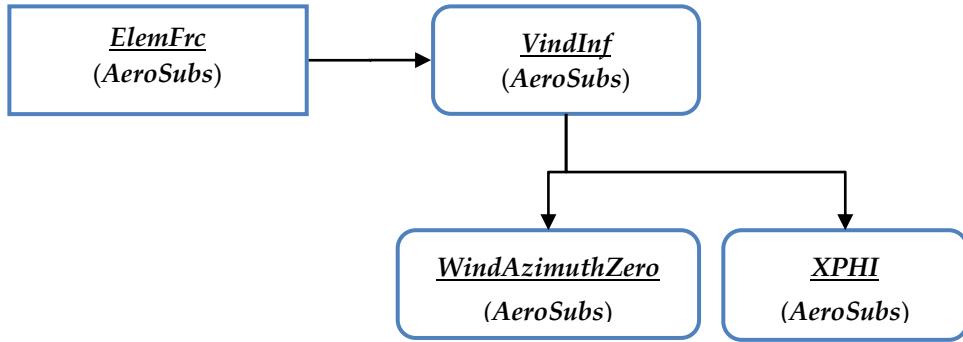


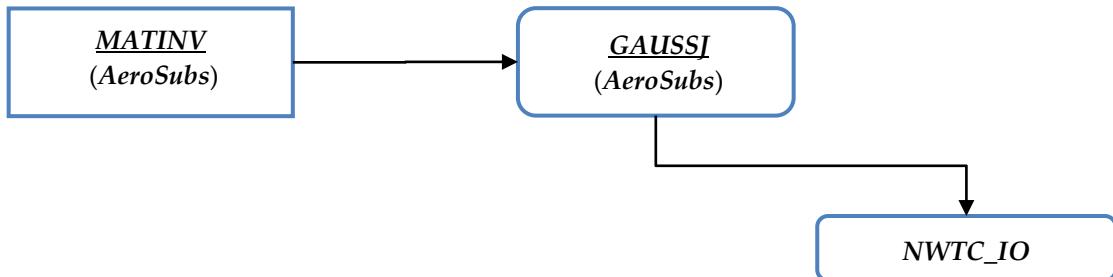
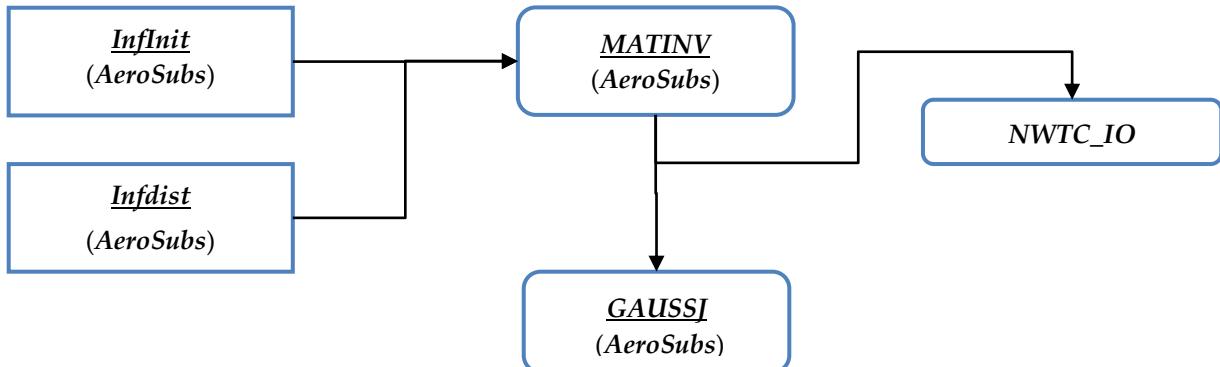
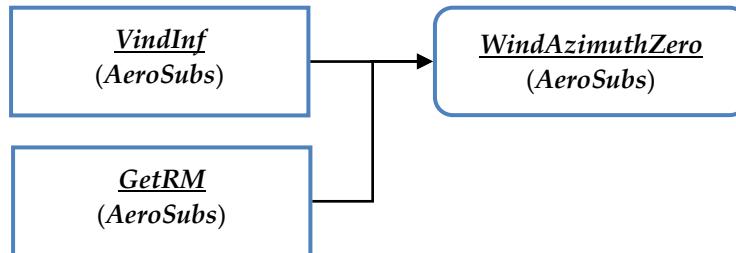
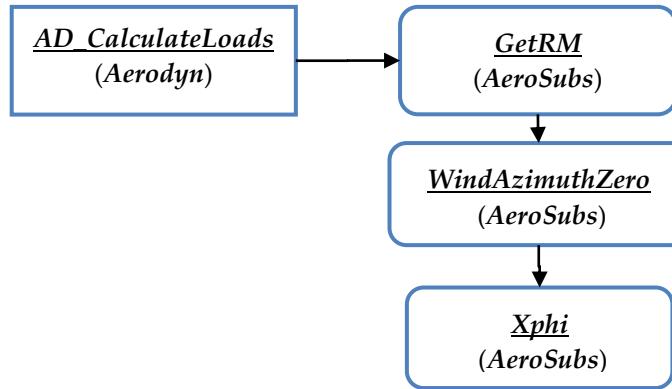


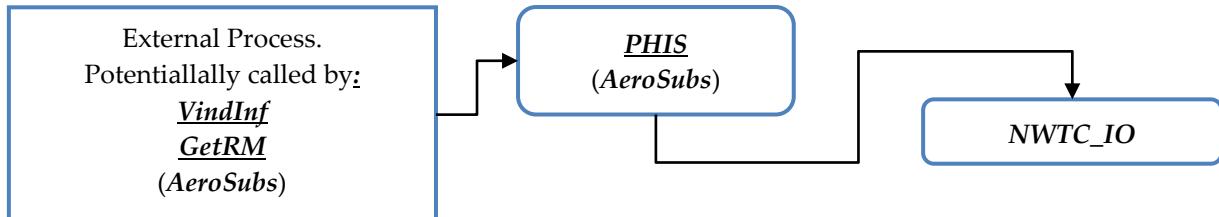
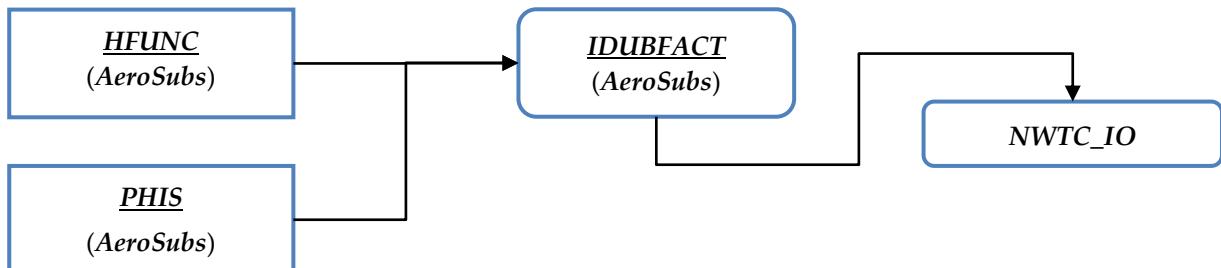
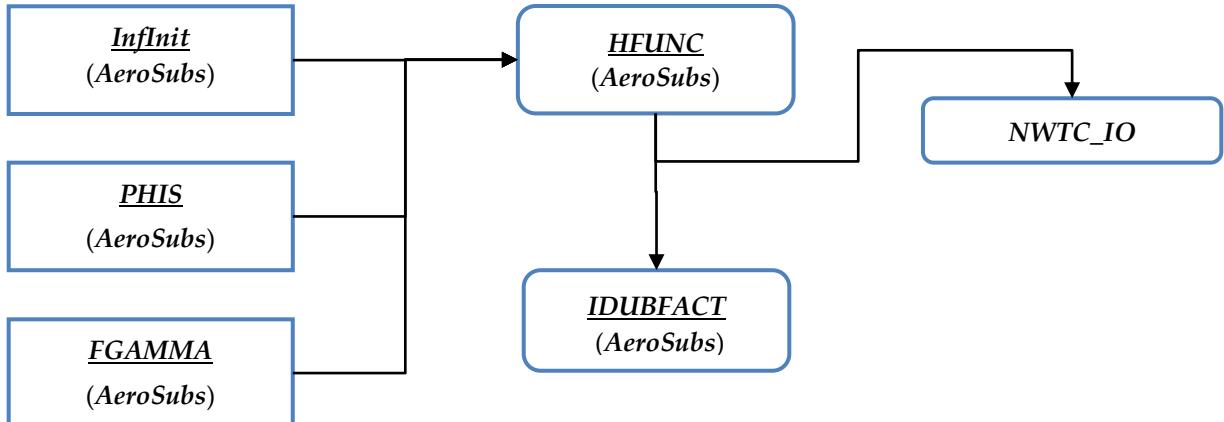


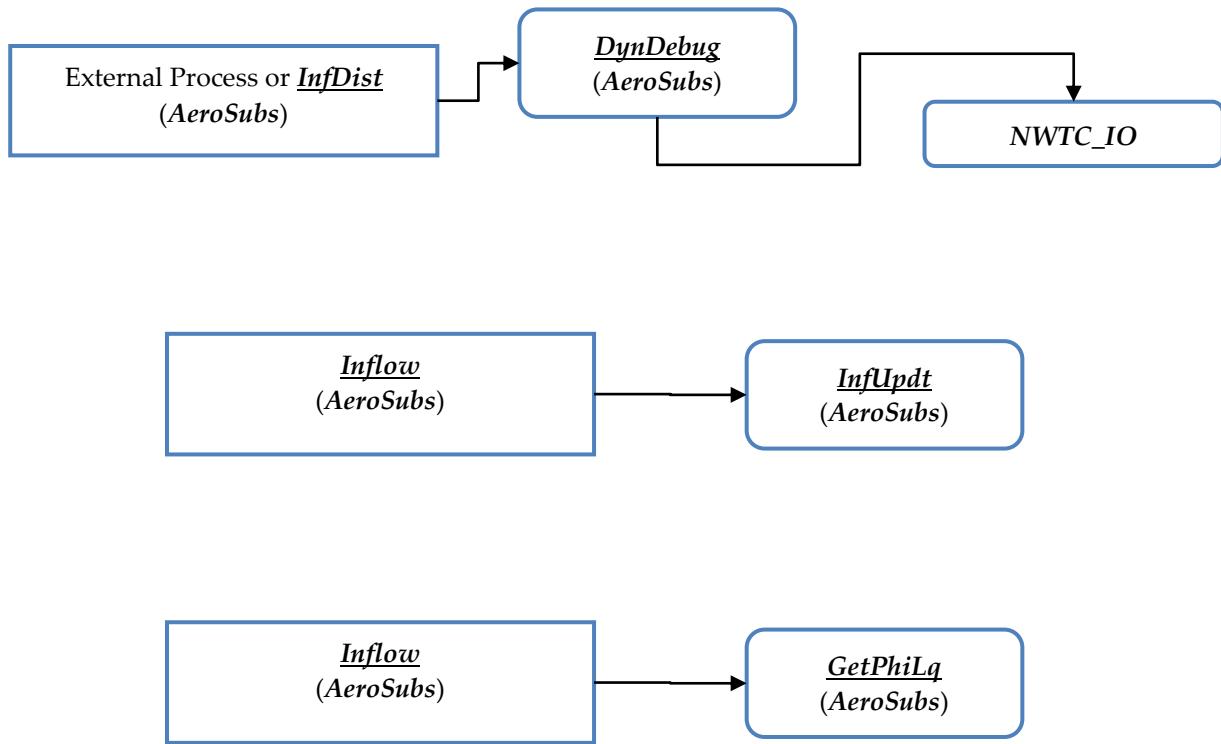




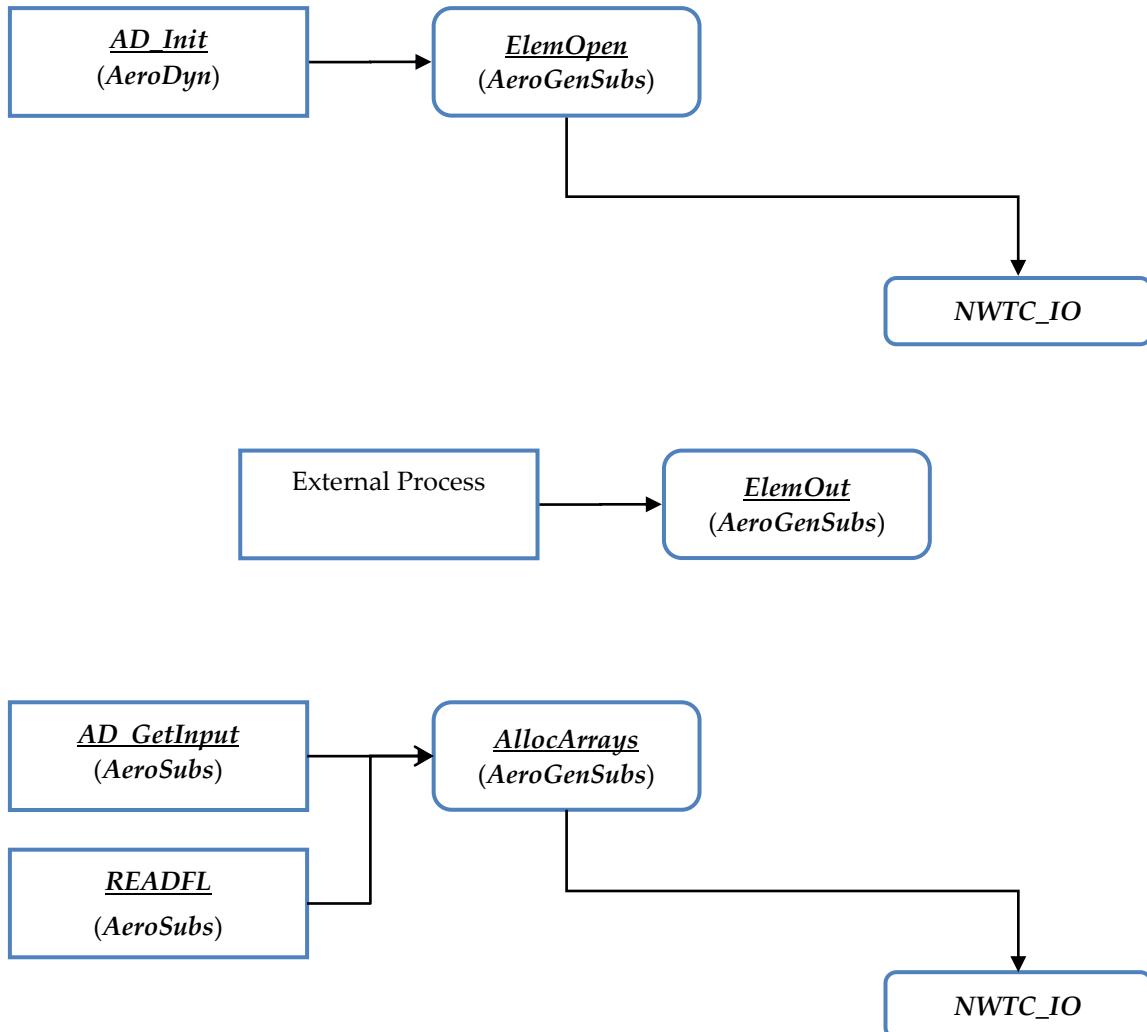




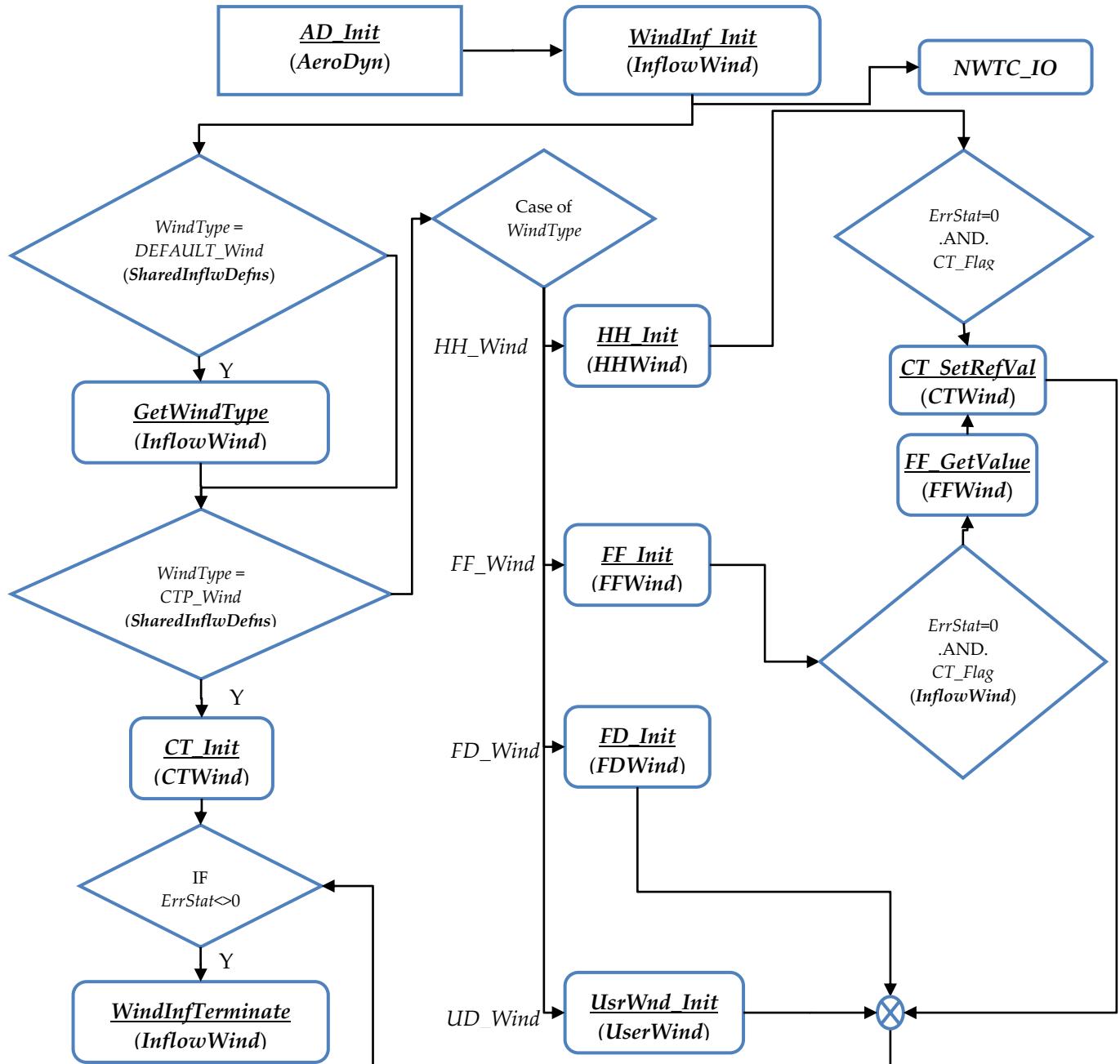


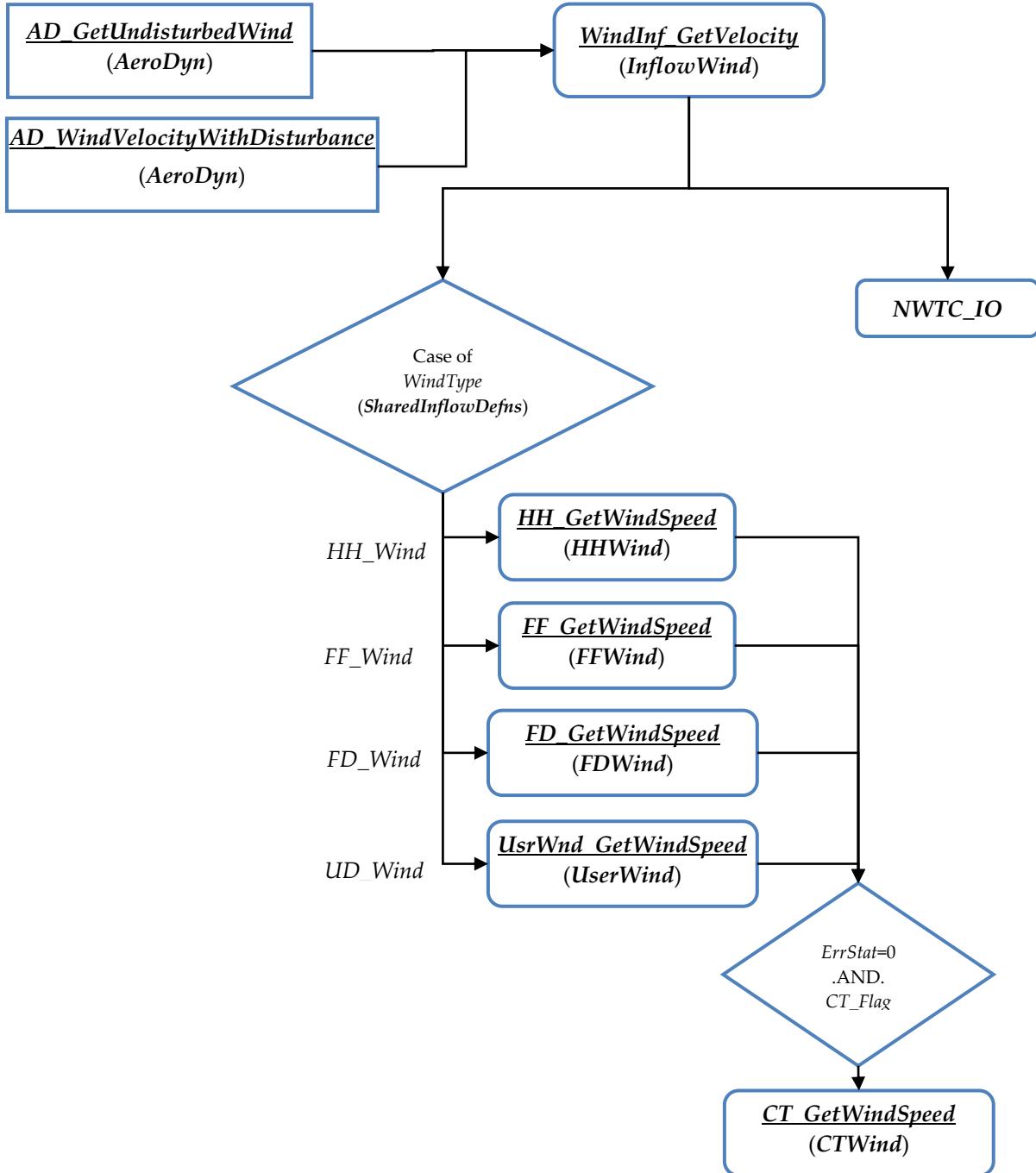


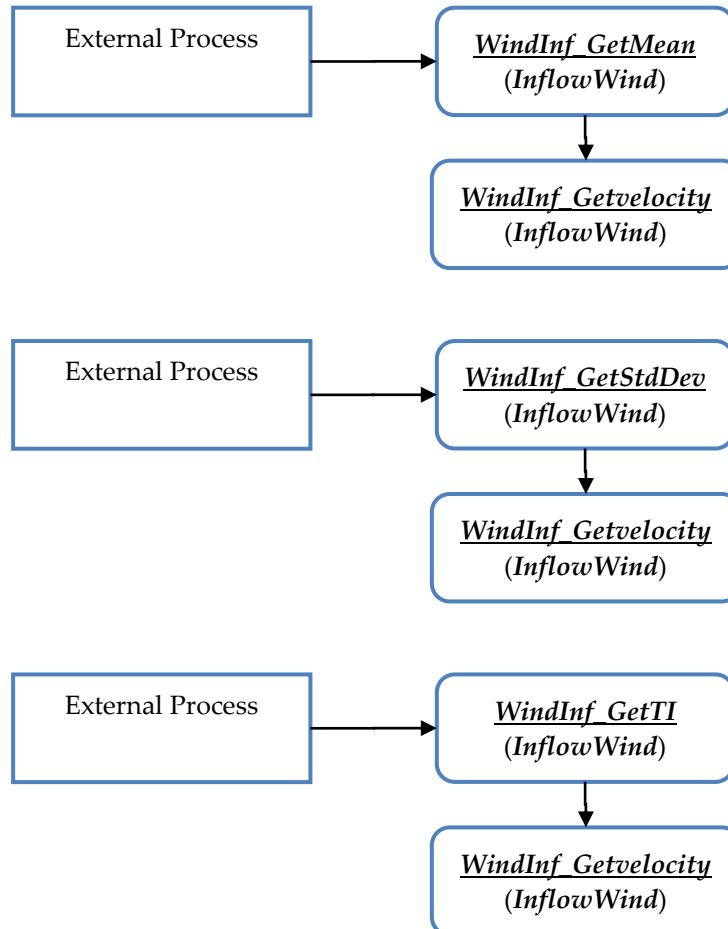
### A3. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE AEROGENSUBS

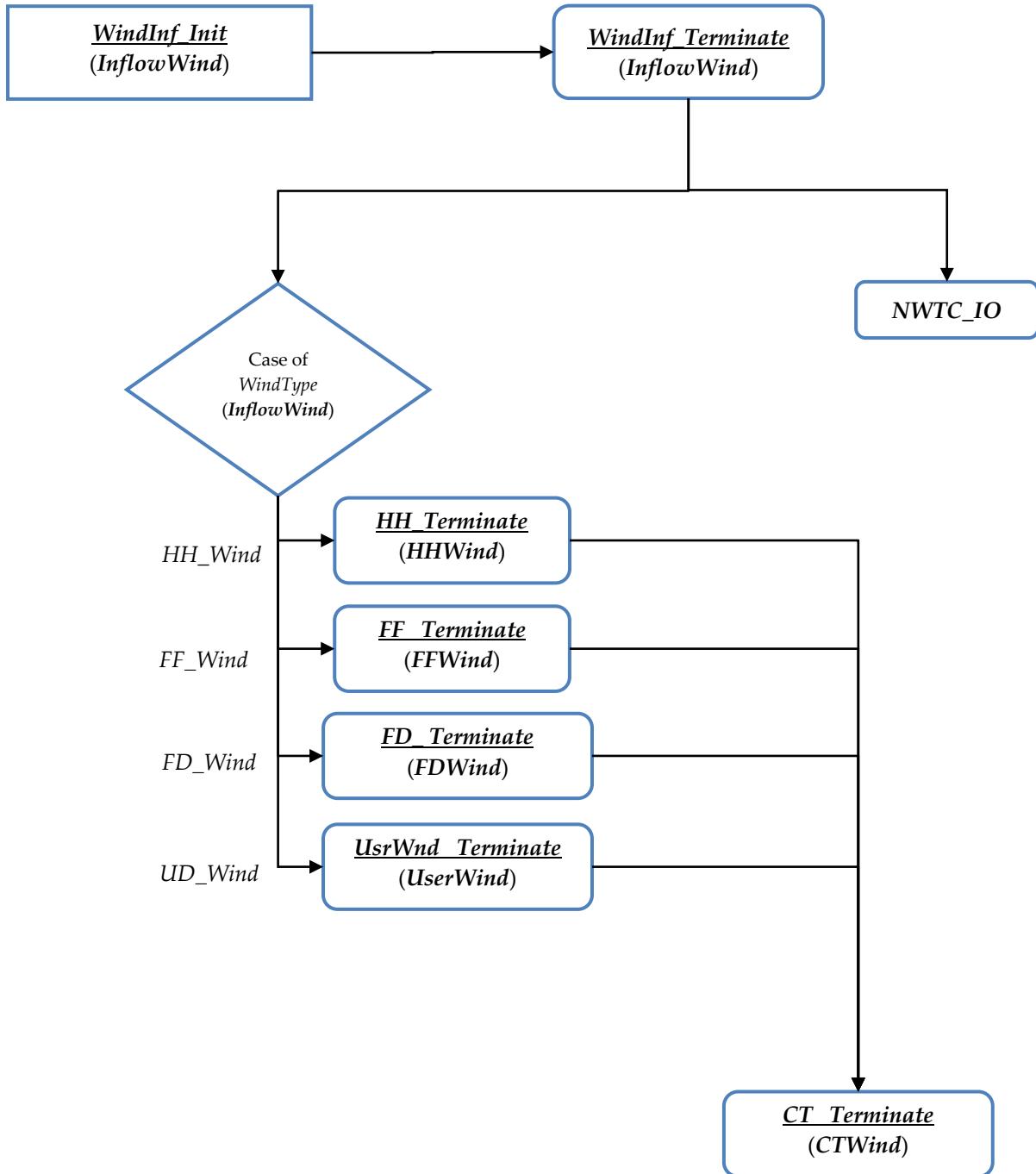


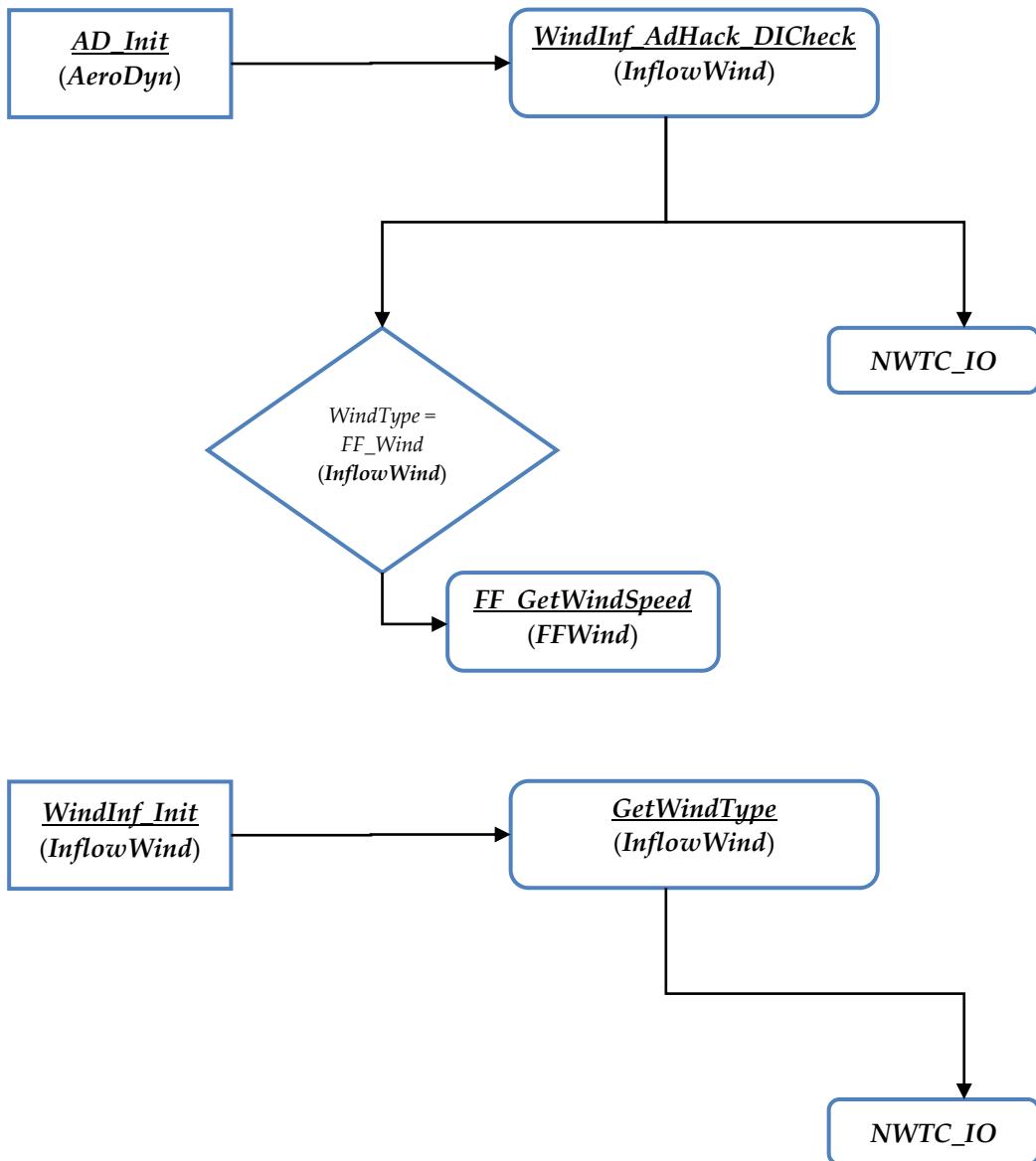
## A4. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE INFLOWWIND

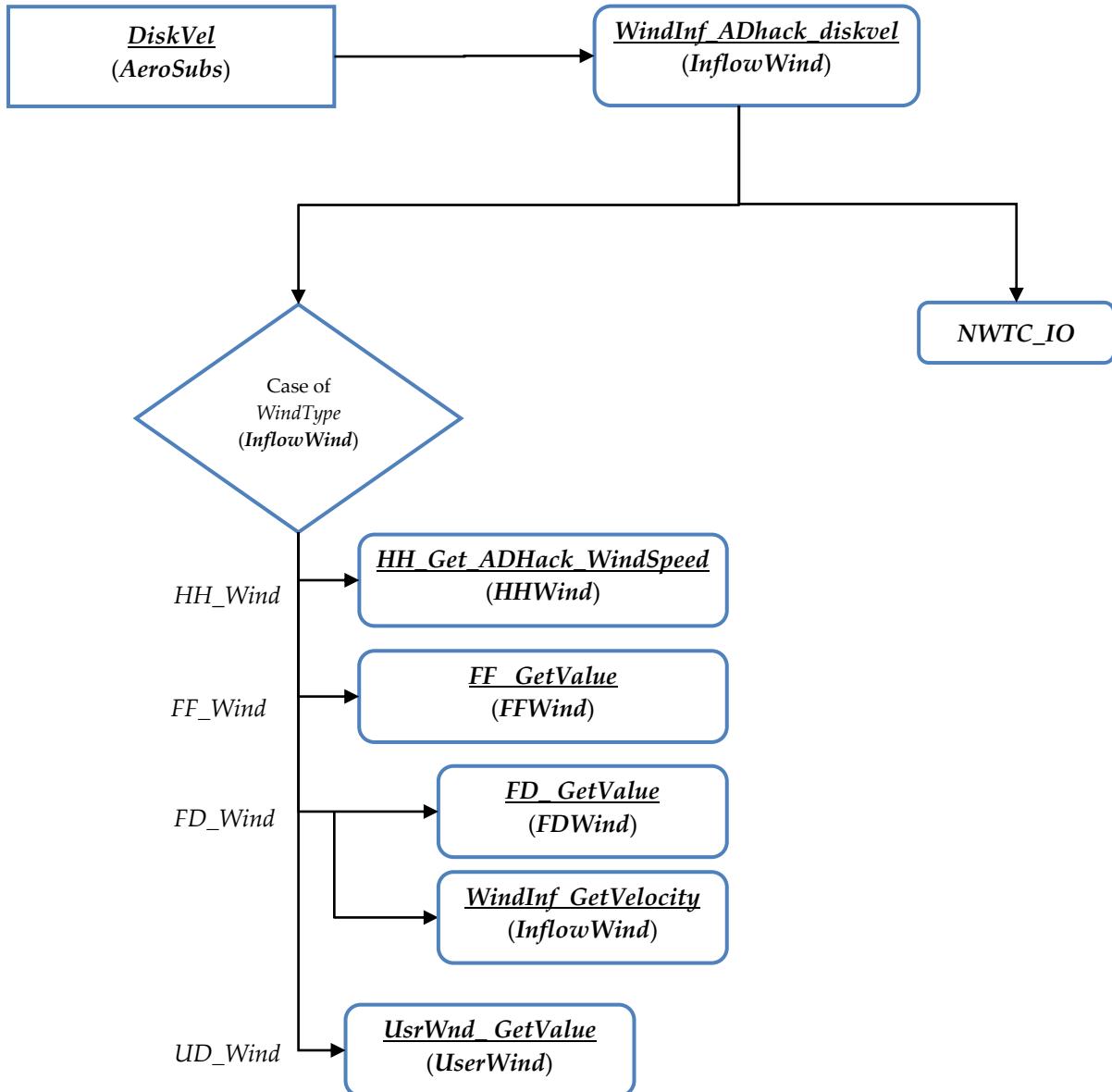


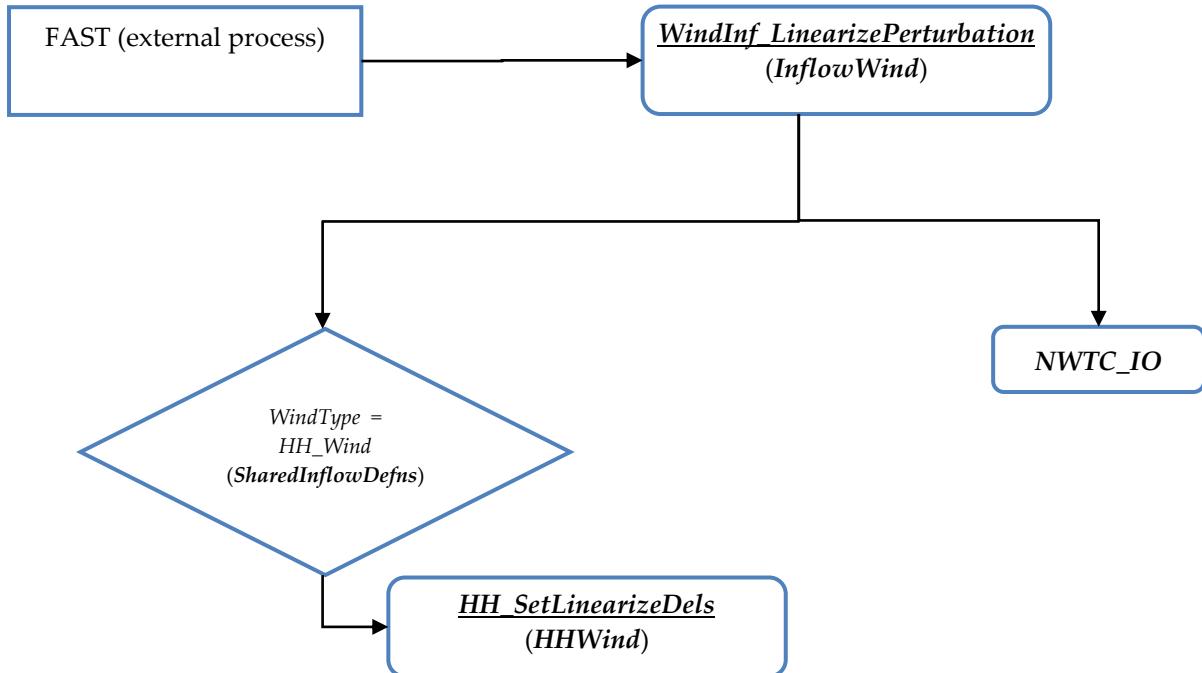




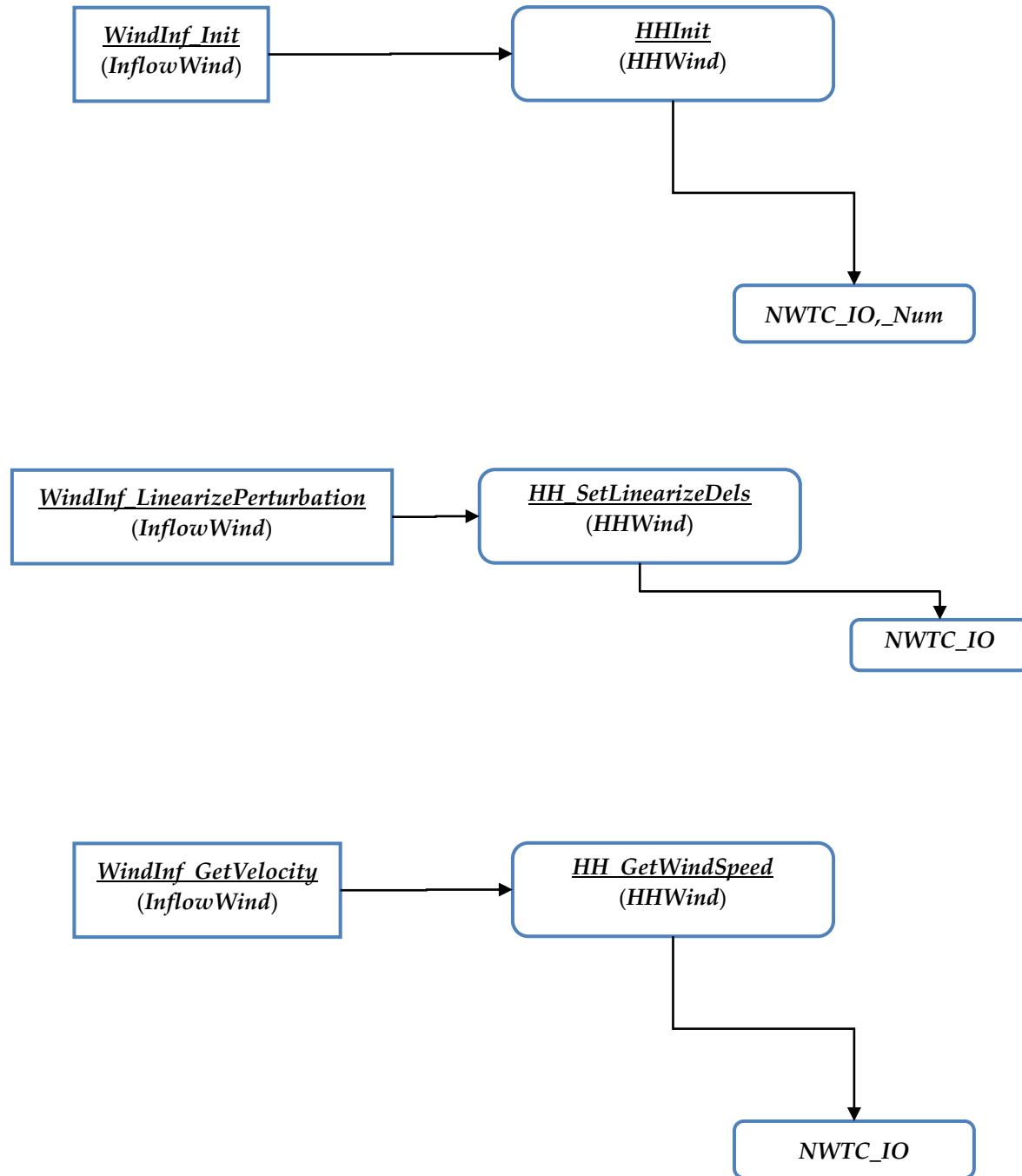


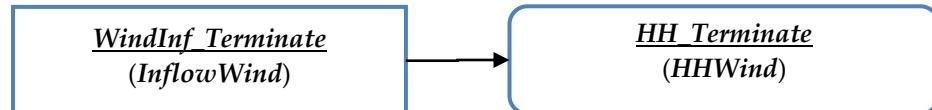
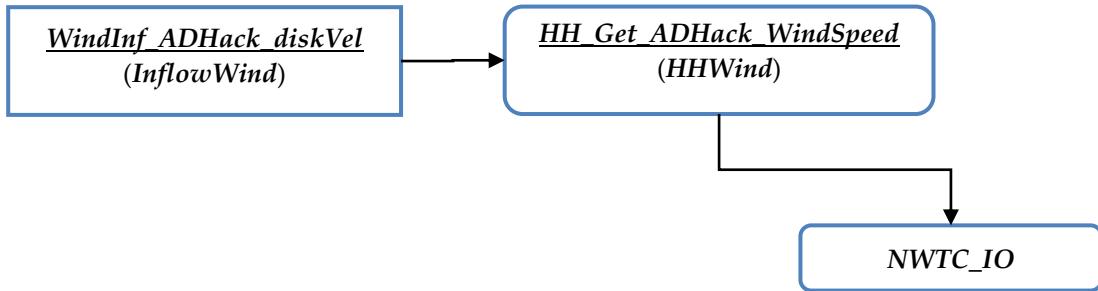




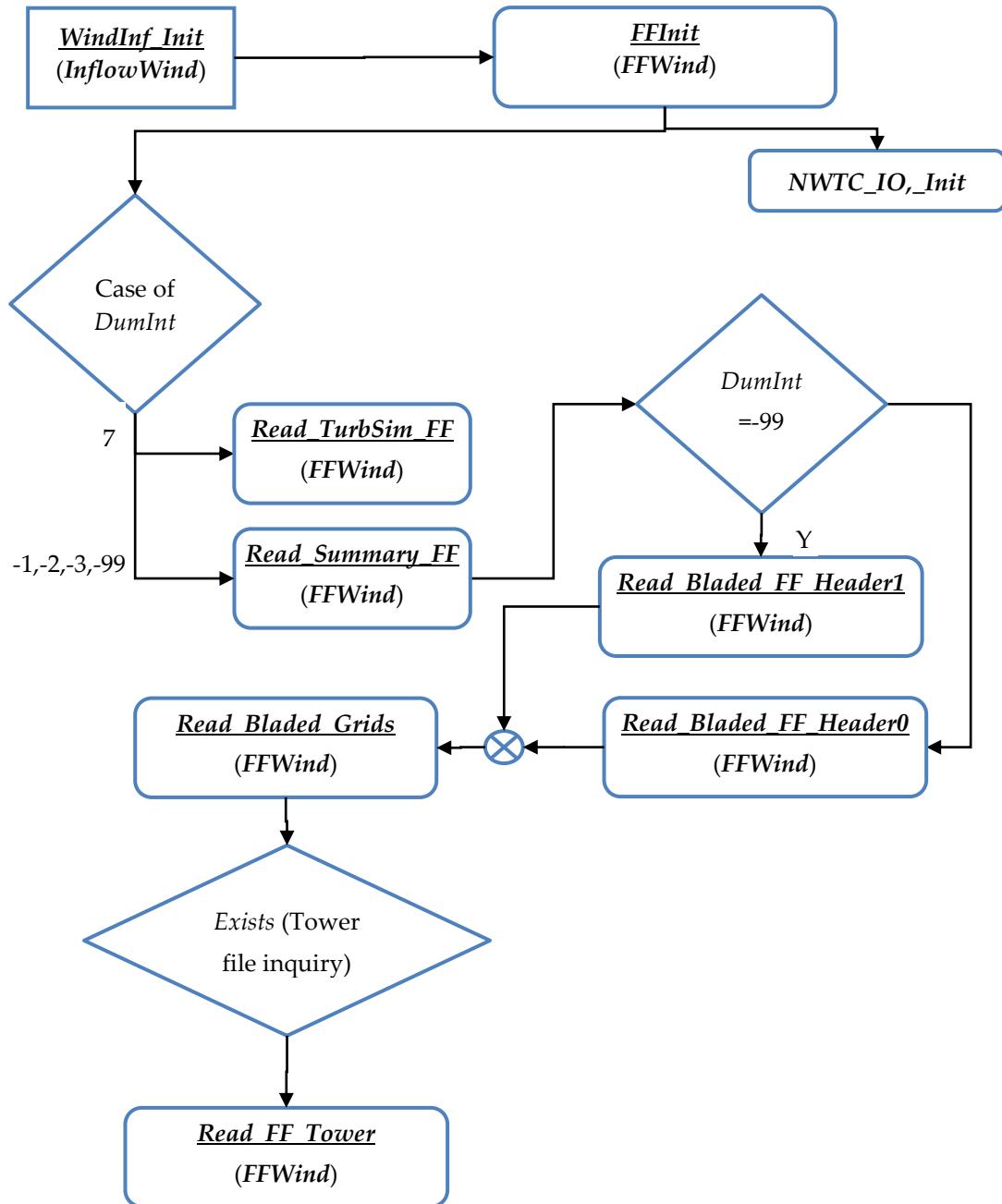


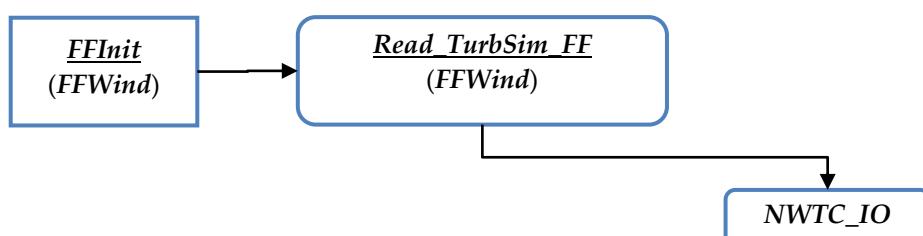
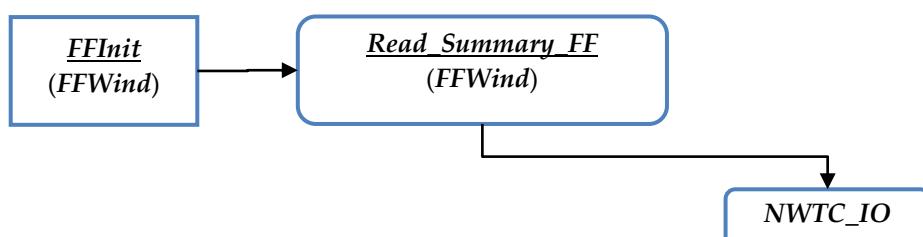
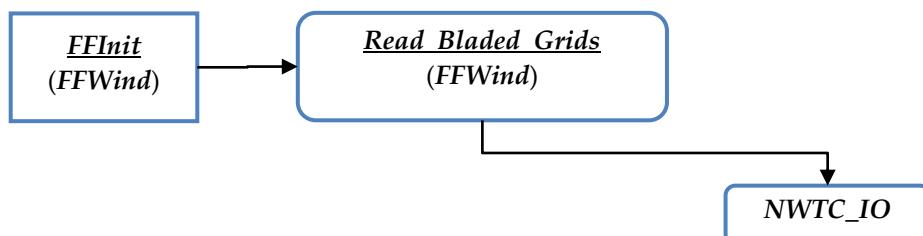
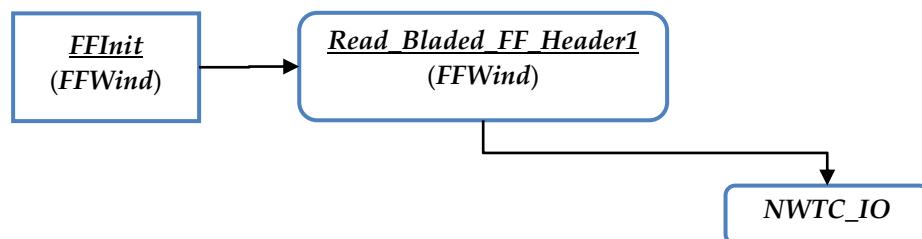
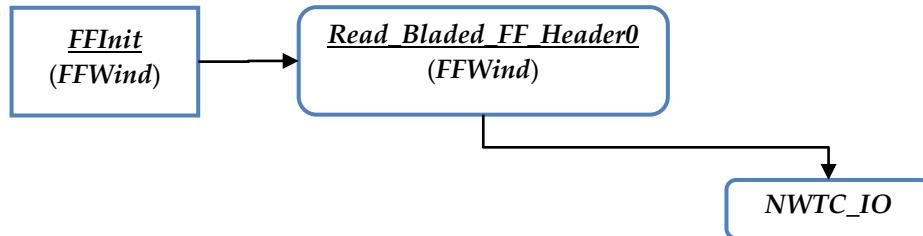
## A5. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE HHWIND

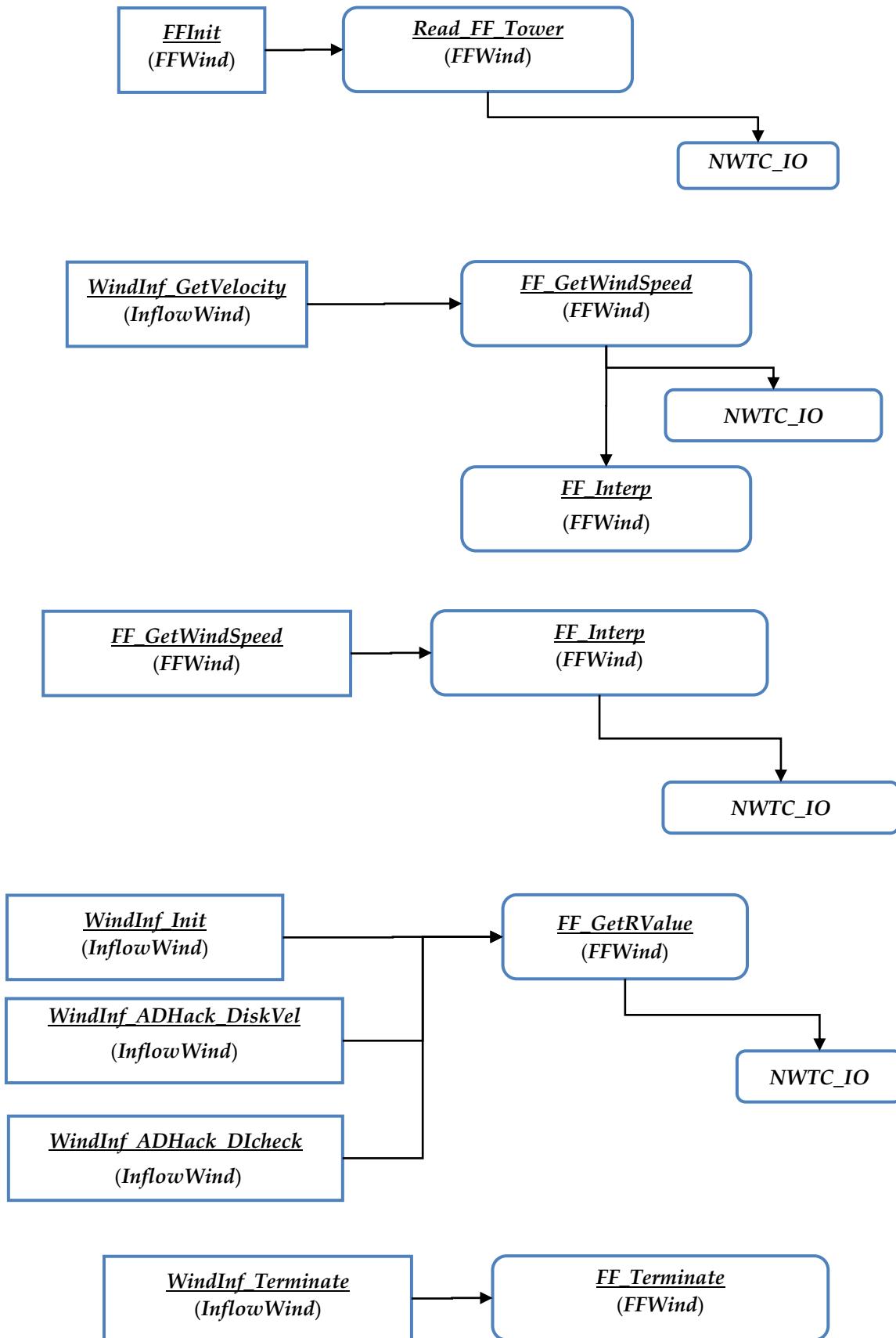




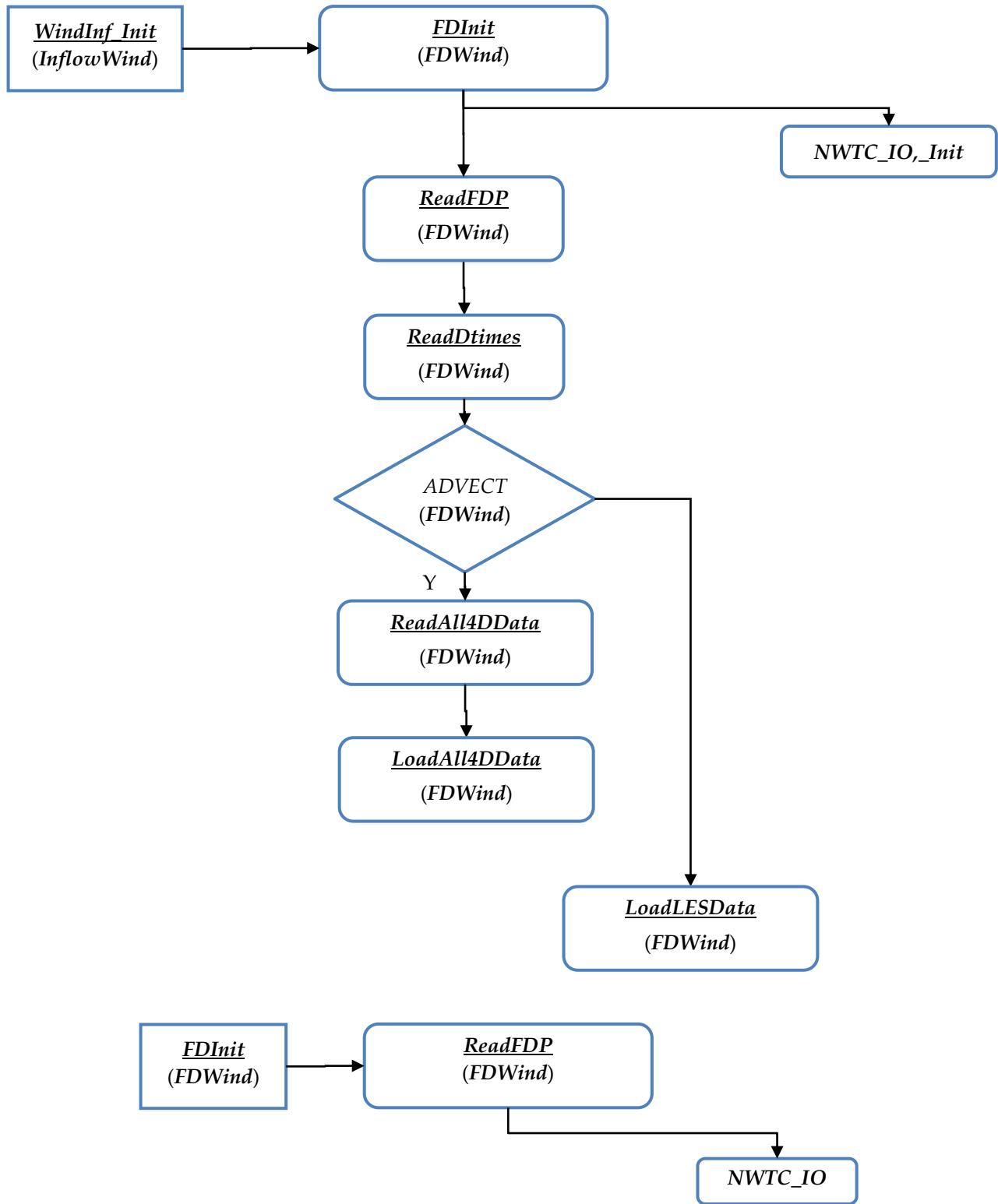
## A6. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE FFWind

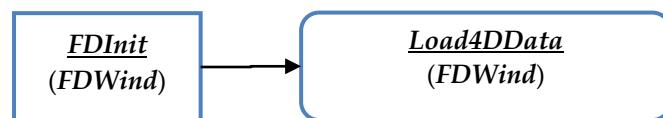
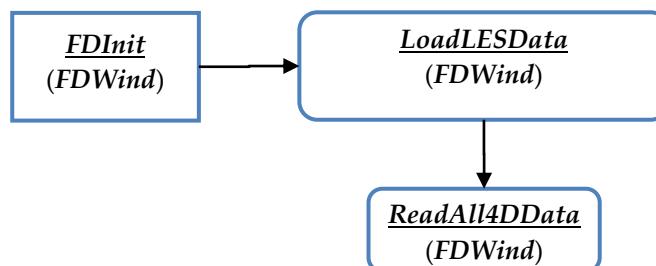
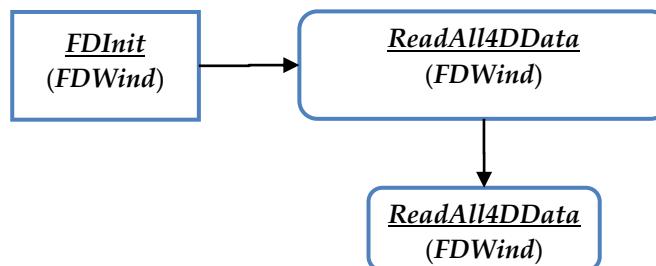
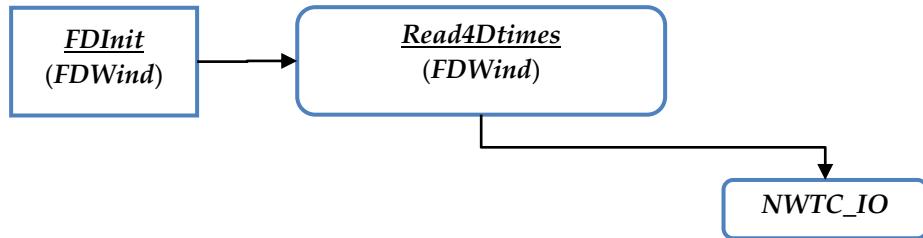


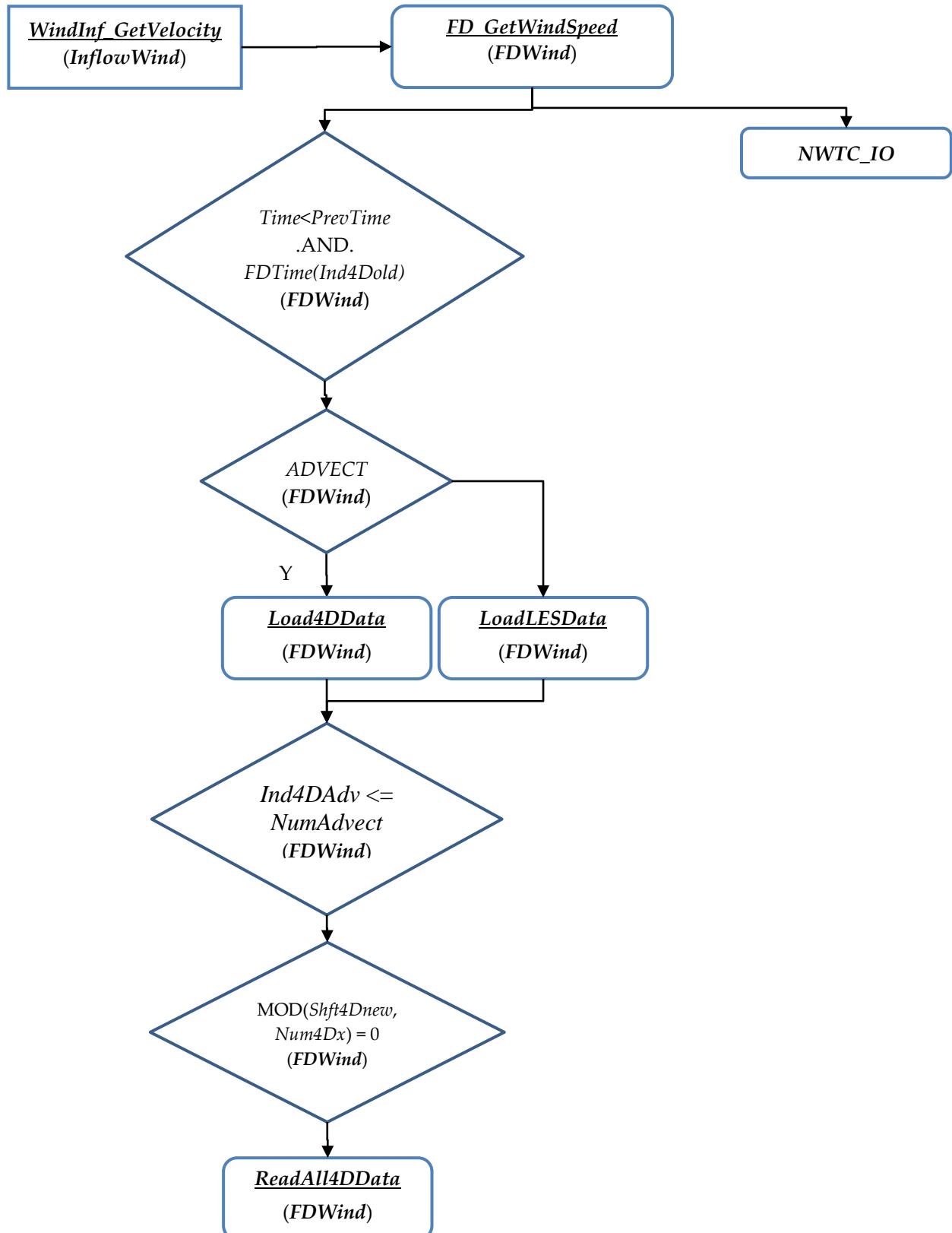


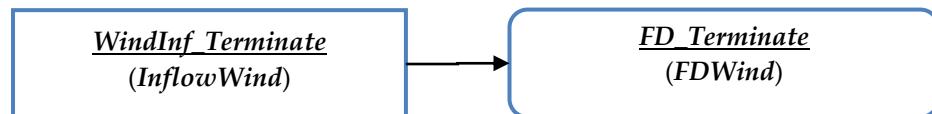
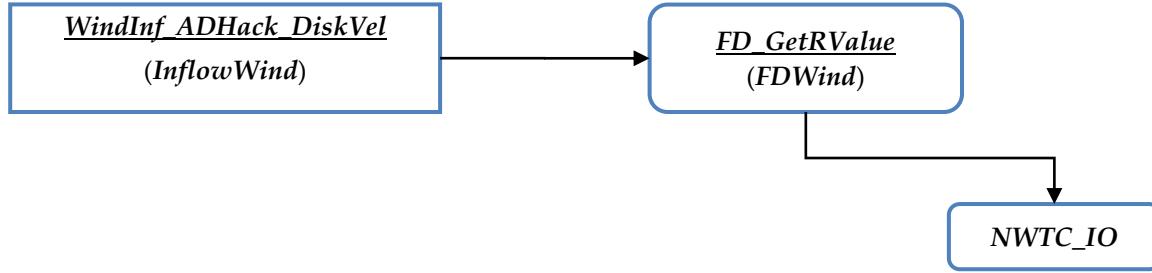


## A7. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE FDWIND

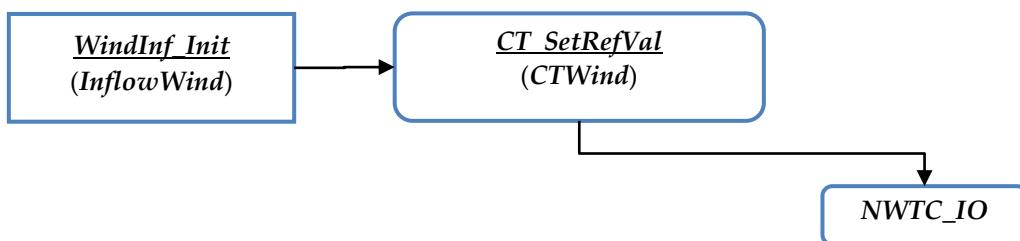
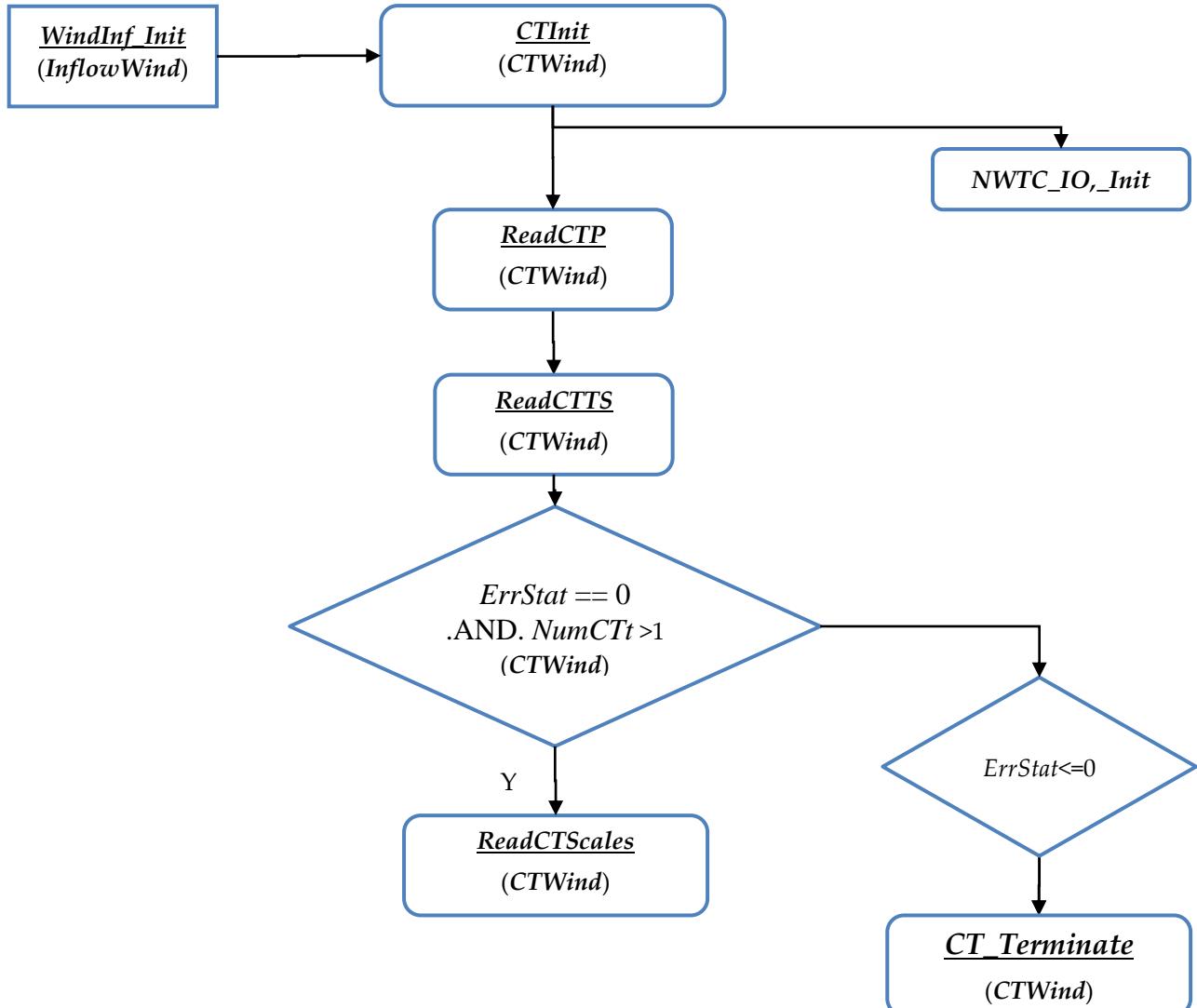


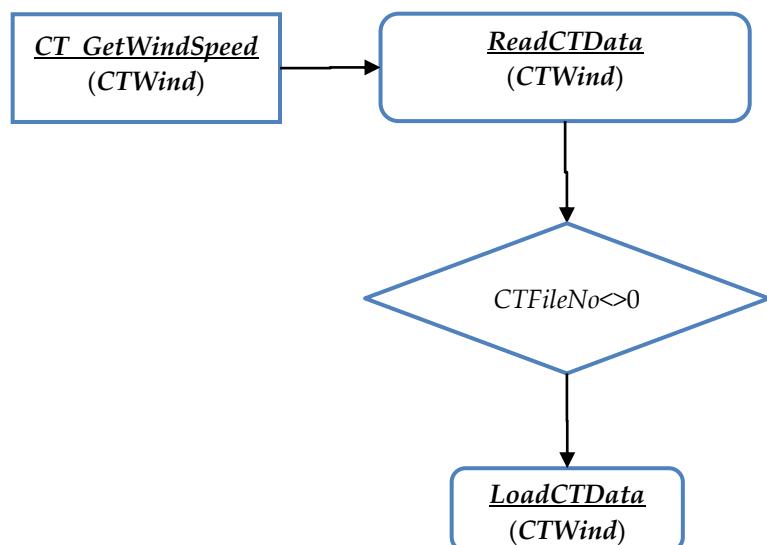
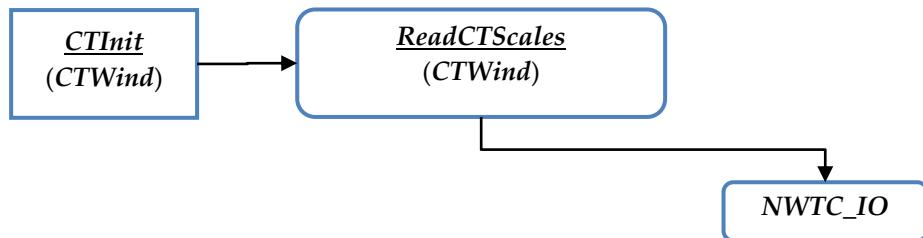
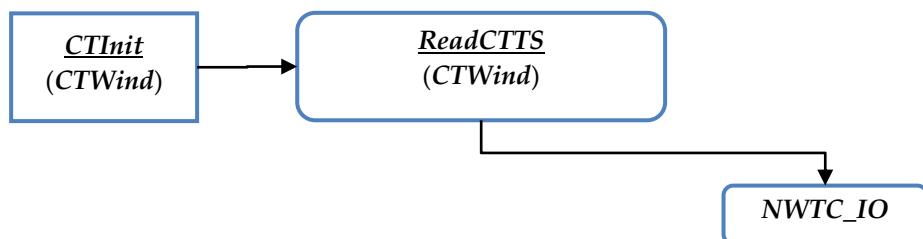
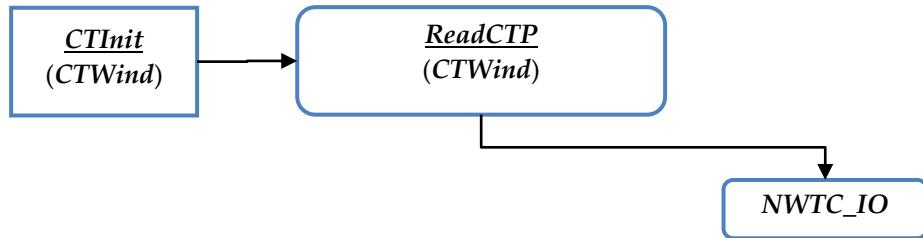


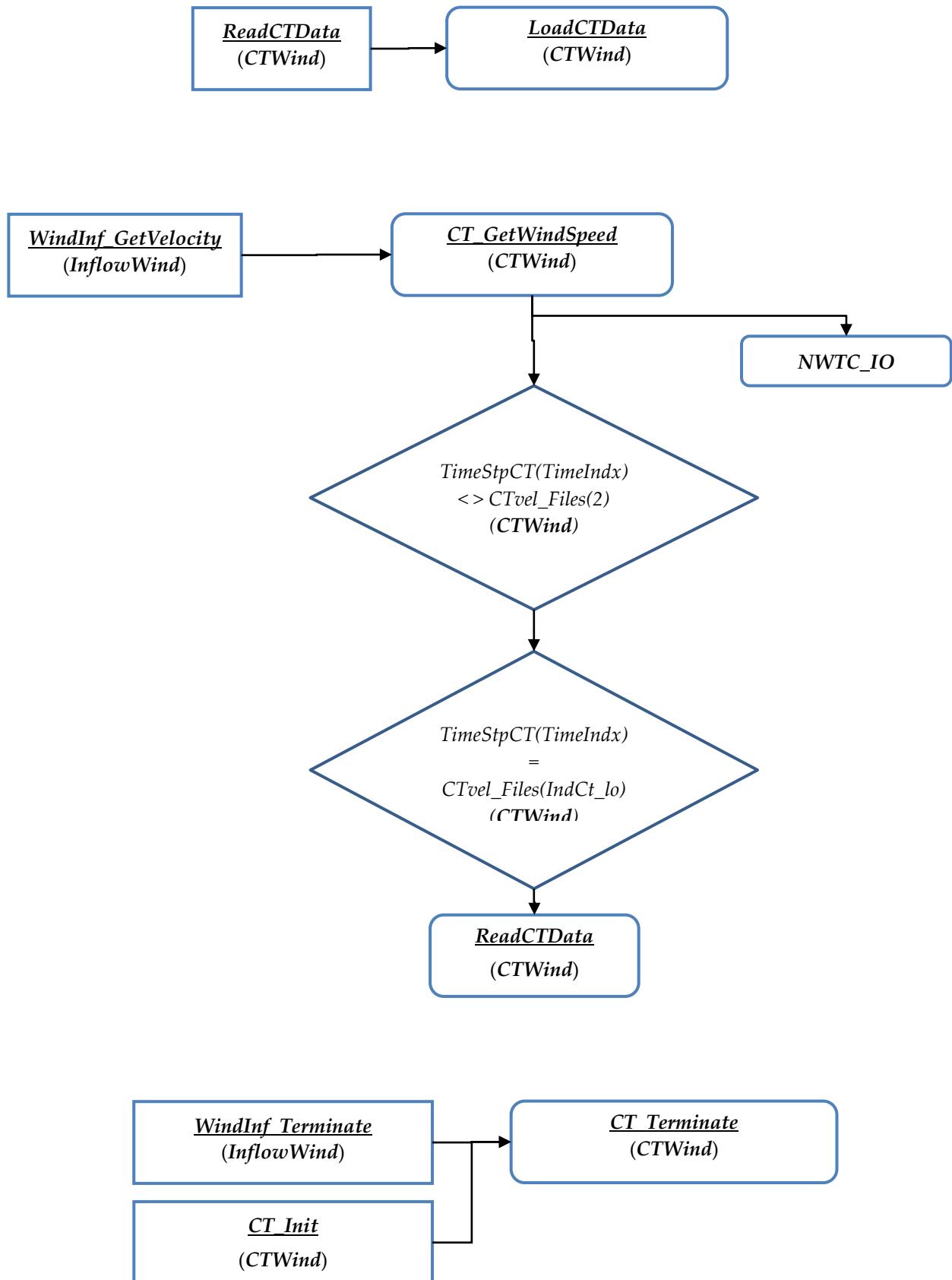




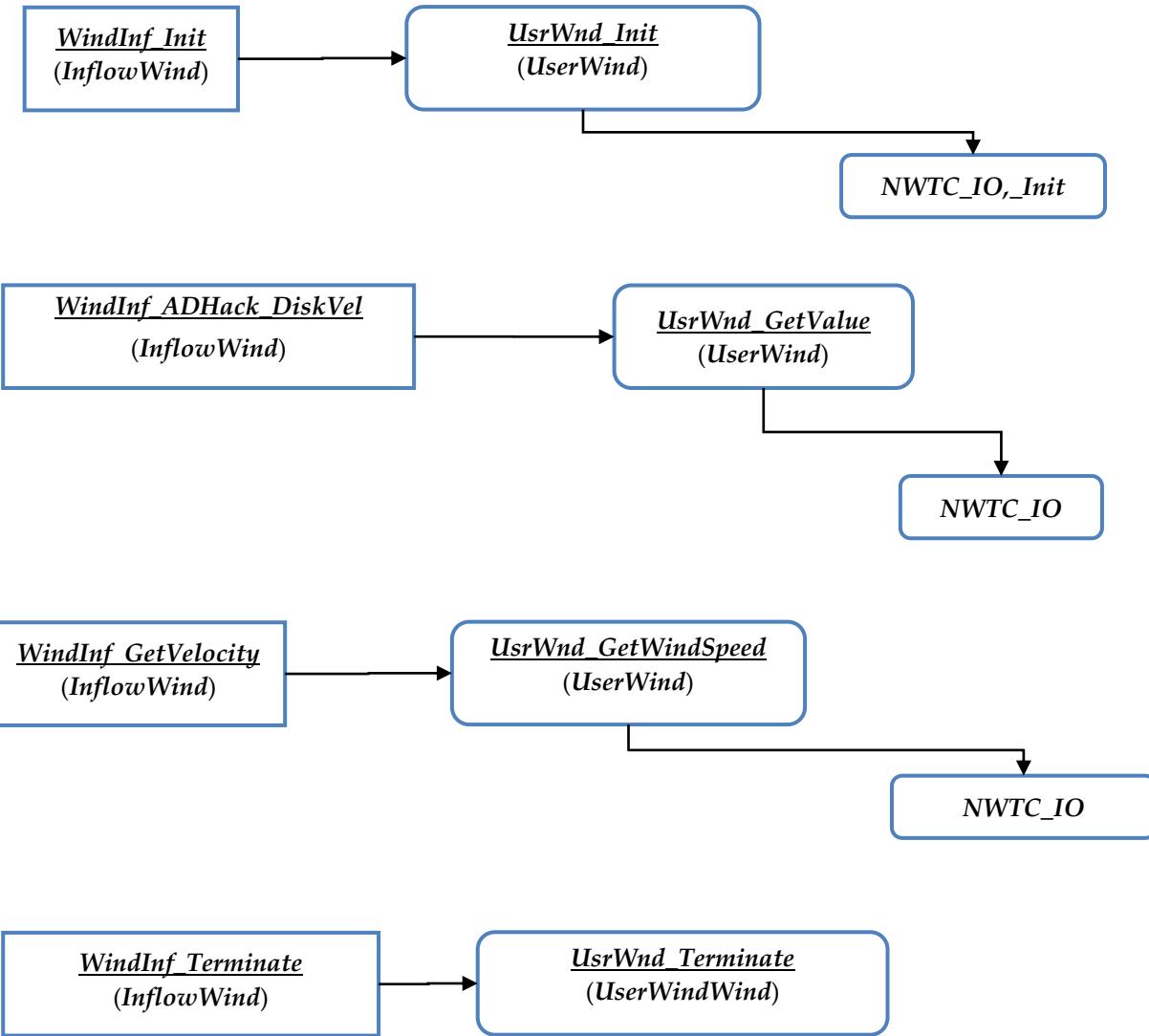
## A8. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE CTWIND







## A9. HIGH-LEVEL FLOWCHARTS FOR ROUTINES AND FUNCTIONS WITHIN MODULE FDWIND



## APPENDIX B. AERODYN MODULE

### GENERAL ORGANIZATION

Aerodyn.f90 contains the module *Aerodyn*, which includes the main interface subroutines between AeroDyn and the driver (structural code). It also contains some new (publically available) types and some local variables.

### EXTERNAL (INVOKED) MODULES

- *NWTC\_Library* [see NWTC\_Library.f90]
- *SharedTypes* [see SharedTypes.f90]
- *InflowWind* [see InflowWindMod.f90]
- *SharedInflowDefs* [see SharedInflowDefs.f90]

### PUBLIC (LOCALLY DECLARED) DATA:

- **AD\_InitOptions:** A new type made up of 2 string(1024) and 1 logical variables

Name	Type	Value	Parameter/ Initialized	Description
<i>ADInputFile</i> (1024)	C			Aerodyn Input File
<i>OutRootName</i> (1024)	C			Root name of summary and element files
<i>WrSumFile</i>	L			Summary File Y/N

- **AeroLoadsOptions:** new type made up of 2 logical and 1 real variables

Name	Type	Value	Parameter/ Initialized	Description
<i>MulTabloc</i> (:,:)	R(ReKi)			$(Nelm(Element),Nb(Blade))$ e.g. Reynolds value
<i>SetMulTabLoc</i> (:,:)	L			$(Nelm(Element),Nb(Blade))$ whether or not MulTabloc is activated for the BE
<i>LinearizeFlag</i>	L			Linearize Y/ N

- A new variable

Name	Type	Value	Parameter/ Initialized	Description
<i>AD_Prog</i> ( <i>SharedTypes</i> )	<b>ProgDesc</b>	ProgDesc('AeroDyn', '(v13.00.00a-bjj, 3jj, 31-Mar-2010)')	Initialized	Prog. version

**FUNCTIONS:**

Name	Description
<u><a href="#">AD_INIT</a></u>	Function to initialize Aerodyn main variables and options
<u><a href="#">AD_CalculateLoads</a></u>	Function to calculate blade element loads
<u><a href="#">AD_GetUndisturbedWind</a></u>	Function that gets pure U-V-W wind velocity (no tower effects)
<u><a href="#">AD_GetCurrentValue</a></u>	Function that returns the scalar value associated with a requested variable name
<u><a href="#">AD_GetConstant</a></u>	Function that returns the scalar value associated with a requested variable name
<u><a href="#">AD_terminate</a></u>	Function that closes files and deallocates variables

**PRIVATE DATA (LOCAL VARIABLES)**

Name	Type	Value	Parameter/ Initialized	Description
<i>Initialized</i>	L (SAVE)	.FALSE.	initialized	Whether or not it is initialized
<i>NoLoadsCalculated</i>	L (SAVE)	.TRUE.	initialized	Whether or not loads have been calculated at the current step
<i>TwoPiNB</i>	R(ReKi)			$2\pi/\text{No. of blades}$
<i>ADCURRENTLOADS</i>	<b>AllAeroLoads (SharedTypes)</b>			copy of current loads to return

## B1. AD\_INIT(ADOPTIONS, TURBINECOMPONENTS, ERRSTAT)

Initialization Function. Called Externally.

### EXTERNAL (INVOKED) MODULES

From the following external modules, AD\_Init extracts definitions of only a few variables

- ***Switch***: [see AeroMods.f90]
  - *ELEMPRN*: logical
  - *DynInfl*: logical; Dynamic inflow switch: .TRUE.=DYNIN vs. .FALSE.=EQUIL
- ***AD\_IOParams***: [see AeroMods.f90]
  - *WrOptFile*: logical
  - *UnADopt*: integer(4)
  - *UnADin*: integer(4)
- ***AeroTime***: [see AeroMods.f90]
  - *Time*: real(DbKi)
  - *OldTime*: real(DbKi)
  - *DTAero*: real(DbKi)
- ***Blade***: [see AeroMods.f90]
  - *NB*: integer(4)
  - *R*: real(*ReKi*)
  - *DR(:)*: real(*ReKi*)
- ***Element***: [see AeroMods.f90]
  - *HLCNST(:)*: real(*ReKi*)
  - *RELM(:)*: real(*ReKi*)
  - *TLCNST(:)*: real(*ReKi*)
  - *TWIST(:)*: real(*ReKi*)
- ***Rotor***: [see AeroMods.f90]
  - *AVGINFL*: real(*ReKi*)
  - *HH*: real(*ReKi*)
- ***InducedVel***: [see AeroMods.f90]
  - *SumInfl*: real(*ReKi*);
- ***ELEMInflow***: [see AeroMods.f90]
  - *ALPHA(:, :)*: real(*ReKi*)
  - *W2(:, :)*: real(*ReKi*)
- ***AeroSubs***: [see AeroSubs.f90]
  - *AD\_GetInput*, *ADOUT*, *CheckRComp* subroutines
- ***AeroGenSubs***: [see GenSubs.f90]
  - *ELEMOpen*: Subroutine

### INPUT

Name	Type	INTENT	Description
<i>ADOptions</i>	<b>AD_InitOptions</b> [see AeroDyn.f90]	IN	Options for Aerodyn
<i>TurbineComponents</i>	<b>AeroConfig</b> [see SharedTypes.f90]	IN	Initial Configuration of Turbine Components
<i>ErrStat</i>	I	OUT	If $\neq 0$ error was encountered

**LOCAL VARIABLES**

Name	Type	Value	Description
<i>CosPrecone</i>	R(ReKi)		Cosine of Precone Angle
<i>DTip</i>	R(ReKi)		To calc.Hub/tip loss constants
<i>ElemRad</i>	R(ReKi)		To calc.Hub/tip loss constants
<i>Dhub</i>	R(ReKi)		To calc.Hub/tip loss constants
<i>Rhub</i>	R(ReKi)		To calc.Hub/tip loss constants
<i>HubRadius</i>	R(ReKi)		Hub radius along blade z axis
<i>MeanWind</i>	R(ReKi)		
<i>TipRadius</i>	R(ReKi)		<i>HubRadius</i> + blade length
<i>TmpVar</i>	R(ReKi)		
<i>IB</i>	I		Blade index counter
<i>Ielm</i>	I		Blade element index counter
<i>InitWindInfl</i>	<b>InflInitInfo</b> [ <i>InflowWind</i> invoked by parent module <i>Aerodyn</i> ] see InfloWindMod.f90]		
<i>Title</i>	C(1024)		

**STEPS**

1. It checks if already initialized: IF *Initialized* THEN exit; ELSE Call **NWTC\_Init** [*NWTC\_Library.f90*] which establishes basic constants (like pi/2 etc.) and opens console for standard output.
2. Outputs message with name and version of program to the screen
3. Gets number of blades defined from input *TurbineComponents*, and the *WrOptFile* (logical).
4. Calls **AD\_GetInput** [see AeroSubs.f90] with some input data and more parameters coming from the used modules, so read in input file and airfoil data files

5. Allocate *W2* and *Alpha* (*ElemInflow* see AeroMods.f90) if not allocated
6. If *ElemPrn* (*Switch*) call *ElemOpen* (*AeroGenSubs*) that will open the element output file (“.elm”) and write main headings
7. Calculate *HubRadius* (along blade z axis) by DOT\_PRODUCT of (*TurbineComponents%Blade(1)%Position(:)* - *TurbineComponents%Hub%Position(:)* , *TurbineComponents%Blade(1)%Orientation(3,:)* ); then repeated for Blade(2) and Blade(3) to check IF it is the same for all blades, ELSE use *HubRadius* calculated from blade 1. Note: Orientation(3,:) are the direction cosines of the blade z-axis w.r.t. the inertial reference frame.
8. Calculate internal variable *TipRadius* = *TurbineComponents%BladeLength* + *HubRadius*
9. Calculate the Precone Angle via a slick way to use ASIN of the DOT\_PRODUCT of the direction cosines of the blade z-axis and hub x-axis [ (ASIN(SIN(90- $\Theta$ )) = ASIN(COS( $\Theta$ )) = ASIN(DOT\_PRODUCT(Z<sub>b</sub>,X<sub>h</sub>)) ], also compare that angle as derived from each blade and in case warn user if it differs. Then calculate *CosPrecone*, i.e. the cosine of the precone angle.
10. Calculate *R* (rotor radius in *Blade*) = *TipRadius* \* *CosPrecone*
11. Calculate *RHub* (internal variable)= *HubRadius* \* *CosPrecone*
12. Call *CheckRComp* (*AeroSubs*) sending *HubRadius* and *TipRadius* in order to verify DRs chosen by user are compatible with the geometry, Else abort
13. Calculate Tip-Loss Constants *TLCNST(:)* (one per blade element based on the distance from the tip)  $0.5*NB * (R - \text{Radius of element}) / \text{Radius of element}$
14. If *RHub*>1 mm, calculate Hub-Loss Constants (one per blade element based on the distance from the hub) as  $0.5*NB * (\text{Radius of element} - RHub) / RHub$ ; else set *HLCNST(:)=0.0*
15. If *WrOptFile* (*AD\_IOParams*), then open the “.opt” file and Call *ADOut* (which also calls *BEDWRT*) and close the “.opt” file
16. Set a few default values for:
  - a. *InitWindInfl%WindFileType*= *DEFAULT\_Wind* (*SharedInflowDefns* invoked by *InflowWind*),
  - b. *InitWindInfl%ReferenceHeight*=*HH* (*Rotor*)
  - c. *InitWindInfl%Width* =*2\*R*
  - d. Note *InitWindInfl% WindFileName* was initialized in *AD\_GetInput* (*AeroSubs*)
17. Call *WindInf\_Init* (*InitWindInfl*, *ErrStat* ) (*InflowWind*)
18. IF *ErrStat*<>0 THEN RETURN
19. IF *DynInfl* (*Switch*) THEN
  - a. Set *MeanWind* = *WindInf\_ADhack\_DLcheck*( *ErrStat* ) (*InflowWind*), i.e. calculate a mean wind speed
  - b. If no errors and *MeanWind* <8.0 then *DynInfl* = .False., i.e. turn off dynamic inflow
20. ALLOCATE (if NOT allocated) current loads return values and *AD\_Init* return values:
  - a. *ADCcurrentLoads%Blade(NElm, NB)*
  - b. *AD\_Init%Blade(NElm, NB)*; type(**AllAeroMarkers**) [see SharedTypes.f90]
21. Set *AD\_Init%Blade(:, :)%Position(1 and 2)=0.0*, i.e. for each element along each blade fix position(1:2) to 0.

22. Set  $AD\_Init\%Blade(:,1:NB)\%Position(3)=RElm(:)$  -  $HubRadius$ ; i.e.: for each element, set the z-coordinate to be equal to the  $RElm(:)$  –  $HubRadius$  (previously calculated). Note  $RElm$  is the distance along z of the element, not the radius of the element due to possible precone.
23. Set  $AD\_Init\%Blade(:,1:NB)\%Orientation(:,::)$  as a function of the  $TWIST$  (direction cosines of the x,y,z local to the blade elements w.r.t. blade frame system
24. Initialize the following variables
  - a.  $SumInfl=0.0$  (**InducedVel**)
  - b.  $AvgInfl=0.0$  (**Rotor**)
  - c.  $Time =0.0$  (**AeroTime**)
  - d.  $OldTime= 0.0$  (**AeroTime**)
  - e.  $TwoPiNB=2*pi/NB$  (parent module **AeroDyn**)
25. Set  $Initialized=.TRUE.$ . (parent module **AeroDyn**) and  $NoLoadsCalculated = .TRUE.$ . (parent module **AeroDyn**)

## OUTPUT

*AD\_Init*: type(**AllAeroMarkers**) [see SharedTypes.f90]

This function return relative position of blade elements  $AD\_Init\%Blade(1:Nelm,1:NB)\%Position(1:3)$  and relative orientation  $AD\_Init\%Blade(1:Nelm,1:NB)\%Orientation(1:3)$ .

It allocates the variable to store current loads  $ADCURRENTLOADS\%Blade(Nelm, NB)$ .

It calls a routine to read wind file, and therefore it fills all the variables (**InflowWind**) relative to wind inflow.

The function also initializes  $Time$ ,  $OldTime$ ,  $AvgInfl$ ,  $SumInfl$ ,  $TwoPiNB$ ,  $Initialized=.TRUE.$  and  $NoLoadsCalculated=.TRUE.$ .

## B2. AD\_GETCONSTANT (VARNAMEERRSTAT)

Function that gets a real scalar whose name is listed in VarName. These are constants, set at the beginning of the program at the time of input file reading or even earlier in modules. Externally Called.

### EXTERNAL (INVOKED) MODULES

From the following external modules, *AD\_GetConstant* extracts definitions of only a few variables

- *AD\_IOParams*: [see AeroMods.f90]
  - *UnADin*: integer(4)
- *AeroTime*: [see AeroMods.f90]
  - *DTAero*: real(DbKi)
- *Rotor*: [see AeroMods.f90]
  - *HH*: real(ReKi)
- *Wind*: [see AeroMods.f90]
  - *KinVisc*: real(ReKi);
  - *RHO*: real(ReKi);

### INPUT

Name	Type	INTENT	Description
<i>VarName</i>	C(*)	IN	Variable Name to be retrieved
<i>ErrStat</i>	I	OUT	If $\neq 0$ error was encountered

Additional Input from invoked module variables

### LOCAL VARIABLES

Name	Type	Value	Description
<i>VarNameUC</i>	C(20)		Variable name in uppcases

### STEPS

1. Set *VarNameUC* as uppercase of *VarName*
2. If 'UNADIN' or 'ADUNIT' set *AD\_GetConstant* = *UnADin* (*AD\_IOParams*) and RETURN (this parameter is known even if aerodyne has not been initialized)
3. Check if *Initialized* (parent module *Aerodyn*), and if not warn to initialize and RETURN. (Other parameters require reading of input file, therefore initialization of Aerodyn)
4. If *VarNameUC* is equal to:
  - a. *REFHT* or *HH* : *AD\_GetConstant*= *HH* (*Rotor*)
  - b. *DT* or *DTAERO* : *AD\_GetConstant*= *DTAero* (*AeroTime*)
  - c. *AIRDENSITY* or *RHO* : *AD\_GetConstant*= *RHO* (*Wind*)
  - d. *KINVISC* : *AD\_GetConstant*= *KinVisc* (*Wind*)

### OUTPUT

*AD\_GetConstant*: R(ReKi)

This function returns certain constant values such as *HH*, *DTAero*, *RHO*, *KinVisc*, *UnADin*.

### B3. AD\_GETCURRENTVALUE (VarName, ErrStat, IBlade, IELEMENT)

Function that gets a real scalar value whose name is listed in VarName. This is a variable that might have been calculated during load calculation for instance, so it is not a constant (parameter).

#### EXTERNAL (INVOKED) MODULES

From the following external modules, AD\_GetCurrentValue extracts definitions of only a few variables

- *Rotor*: [see AeroMods.f90]
  - *AvgInfl*: real(ReKi)
- *ElementInflow*: [see AeroMods.f90]
  - *W2*(*:,:*): real(ReKi);
  - *ALPHA*(*:,:*): real(ReKi);

#### INPUT

Name	Type	INTENT	Description
<i>VarName</i>	C(*)	IN	Variable Name to be retrieved
<i>ErrStat</i>	I	OUT	If $\neq 0$ error was encountered
<i>IBlade</i>	I	IN (OPTIONAL)	Index referring to blade number
<i>IElement</i>	I	IN (OPTIONAL)	Index referring to element number

Additional Input from invoked module variables

#### LOCAL VARIABLES

Name	Type	Value	Description
<i>VarNameUC</i>	C(20)		Variable name in uppcases

#### STEPS

1. Set *VarNameUC* as uppercase of *VarName*
2. If *VarNameUC* is equal to:
  - a. AVGINFL or AVGINFLOW: *AD\_GetCurrentValue*= *AvgInfl* (*Rotor*)
  - b. W2: if IBlade and IELEMENT are present, then *AD\_GetCurrentValue*= *W2*(*IElement, Iblade*) (*Rotor*)
  - c. ALPHA: if IBlade and IELEMENT are present, then *AD\_GetCurrentValue*= *ALPHA*(*IElement, Iblade*) (*Rotor*)
  - d. Else warn the user and RETURN

#### OUTPUT

*AD\_GetCurrentValue*: R(ReKi). This function returns certain variable values such as *AvgInfl*, *W2*, *ALPHA*.

## B4. AD\_GETUNDISTURBEDWIND (TIME, INPUTPOSITION,ERRSTAT)

Function that returns the U-V-W wind speeds at the specified time and X-Y-Z location. Nothing (within Aerodyn) calls this routine: Externally Called.

### EXTERNAL (INVOKED) MODULES

N/A

### INPUT

Name	Type	INTENT	Description
<i>Time</i>	R(ReKi)	IN	Variable Name to be retrieved
<i>InputPosition(3)</i>	R(ReKi)	IN	Position X-Y-Z
<i>ErrStat</i>	I	OUT	If $\neq 0$ error was encountered

Additional Input from parent module (*Aerodyn*) variables

### LOCAL VARIABLES

Name	Type	Value	Description
<i>InflowVel</i>	<i>InflIntrpOut</i> ( <i>SharedInflowDefs</i> ) [used by <i>InflowWind</i> module. See <i>SharedInflowDefs.f90</i> ]		Undisturbed Velocity

### STEPS

1. Set *InflowVel* = WindInf\_GetVelocity( *Time*, *InputPosition*, *ErrStat* ) (*InflowWind*)
2. IF *ErrStat*  $\neq 0$  Call ProgWarn (*NWTC\_IO*), and warn user
3. Set Output *AD\_GetUndisturbedWind*(:) = *InflowVel*%*Velocity*(:)

### OUTPUT

*AD\_GetUndisturbedWind*(3): Array R(ReKi)

This function returns the U-V-W wind speeds at specified time and coordinates.

## B5. AD\_TERMINATE (ERRSTAT)

Subroutine that deallocates variables and closes files. Externally Called.

### EXTERNAL (INVOKED) MODULES

- *ElemOutParams*: [see AeroMods.f90]
  - *UnElem*: I(4)
  - *UnWndOut*: I(4)
- *AeroSubs*: [see AeroSubs.f90]

### INPUT

Name	Type	INTENT	Description
<i>ErrStat</i>	I	OUT	If $\neq 0$ error was encountered

Additional Input from parent module (*Aerodyn*) variables.

### LOCAL VARIABLES

N/A

### STEPS

1. Call WindInf Terminate(*ErrStat*) (*InflowWind*): this routine cleans up wind related variables and files
2. Call AeroDyn Terminate() (*AeroSubs*): this routine deallocates several variables belonging to modules under *AeroMods* and closes file units belonging to modules in *AeroMods*
3. DEALLOCATE *ADCurrentLoads%Blade(NElm, NB)*
4. CLOSE the following file units
  - a. *UnElem* (*ElOutParams*) ("elm", element output file)
  - b. *UnWndOut* (*ElOutParams*) (wind output file)
5. CALL CloseEchoQ(NWTC\_Library) which closes file unit *UnEc* (*NWTC\_Library*) which was already taken care by AeroDyn Terminate (possible minor bug)
6. Reset initialization flags:
  - a. *Initialized = .FALSE.* (*AeroDyn*)
  - b. *NoLoadsCalculated = .TRUE.* (*AeroDyn*)
7. Set *ErrStat=0*

### OUTPUT

This routine is called at the end of aerodyn execution and cleans up all the allocated arrays and closes all files related to aerodyn.

## B6. AD\_CALCULATELOADS(CURRENTTIME, INPUTMARKERS, TURBINECOMPONENTS, CURRENTADOPTIONS, ERRSTAT)

Main Aerodyn Function that calculates loads on the blade elements given by InputMarkers. Externally Called.

### EXTERNAL (INVOKED) MODULES

From the following external modules, *AD\_CalculateLoads* extracts definitions of only a few variables

- *AeroTime* [see AeroMods.f90]
- *Airfoil*: [see AeroMods.f90]
  - *MulTabLoc*: R(ReKi)
- *Blade*: [see AeroMods.f90]
  - *R*: R(ReKi)
  - *DR(:)*: R(ReKi)
  - *NB*: I(4)
- *Element*: [see AeroMods.f90]
  - *PitNow*: R(ReKi)
  - *NElm*: R(ReKi)
  - *TWIST(:)*: R(ReKi)
- *Rotor*: [see AeroMods.f90]
  - *AvgInfl*: R(ReKi)
  - *Tilt*: R(ReKi)
  - *YawAng*: R(ReKi)
  - *Yawvel*: R(ReKi)
  - *Revs*: R(ReKi)
  - *CTilt*: R(ReKi)
  - *CYaw*: R(ReKi)
  - *Stilt*: R(ReKi)
  - *SYaw*: R(ReKi)
- *Switch*: [see AeroMods.f90]
  - *DStall*: logical
  - *Wake*: logical
  - *DynInfl*: logical
  - *DynInit*: logical
  - *ElemPrn*: logical
- *InducedVel*: [see AeroMods.f90]
  - *SumInfl*: R(ReKi)
- *ElOutParams*: [see AeroMods.f90]
  - *SaveVX(:, :)*: R(ReKi)
  - *SaveVY(:, :)*: R(ReKi)
  - *SaveVZ(:, :)*: R(ReKi)

- $VXSAV(:,:,:) : R(ReKi)$
- $VYSAV(:,:,:) : R(ReKi)$
- $VZSAV(:,:,:) : R(ReKi)$
- $WndElPrList(:) : I(4)$
- *AeroSubs*: [see AeroSubs.f90]

**INPUT**

Name	Type	INTENT	Description
<i>CurrentTime</i>	R(ReKi)	IN	Current simulation time
<i>InputMarkers</i>	<b>AllAeroMarkers</b> <i>(SharedTypes)</i>	IN	The input state of all aerodynamic markers
<i>CurrentADOptions</i>	<b>AeroLoadsOptions</b> (parent module <i>Aerodyn</i> )	IN	See <b>AeroLoadsOptions</b>
<i>TurbineComponents</i>	<b>AeroConfig</b> <i>(SharedTypes)</i>	IN	The markers defining the current location of the turbine
<i>ErrStat</i>	I	OUT	If $\neq 0$ , error was encountered

Additional Input from invoked module variables and parent module variables (e.g.: *ADCURRENTLOADS*).

**LOCAL VARIABLES**

Name	Type	Value	Parameter/ Initialized	Description
<i>OnePlusEpsilon</i>	R(ReKi)	$1 + \text{EPSILON}(\text{CurrentTime})$	PARAMETER	Minimum computer number $> 1$
<i>VNElement</i>	R(ReKi)			Component normal to the plane of rotation of deflection translational velocity alone
<i>VelNormalToRotor2</i>	R(ReKi)			Square of the wind velocity component normal to the rotor
<i>VNWind</i>	R(ReKi)			Component normal to the plane of rotation of wind velocity alone

$VTT_{Total}$	R(ReKi)	Component in the plane of rotation of (wind – Deflection translational velocity )
$DFN$	R(ReKi)	Elemental Normal Force
$DFT$	R(ReKi)	Elemental Tangential Force
$PMA$	R(ReKi)	Elemental Aerodynamic Moment
$SPitch$	R(ReKi)	Sin(Local Pitch)
$CPitch$	R(ReKi)	Cos(Local Pitch)
$AvgVelNacelleRotorFurlYaw$	R(ReKi)	Relative rotational velocity of the rotor w.r.t. nacelle (z component)
$AvgVelTowerBaseNacelleYaw$	R(ReKi)	Relative rotational velocity of the nacelle w.r.t. tower (z component)
$AvgVelTowerBaseYaw$	R(ReKi)	#3 component of the tower rotational velocity
$AzimuthAngle$	R(ReKi)	Blade azimuth
$rNacelleHub(2)$	R(ReKi)	Horizontal vector from nacelle center (frame-of-reference origin) to hub center (frame-of-reference origin)
$rLocal$	R(ReKi)	Radial distance between hub and blade element, in the plane of rotation
$rRotorFurlHub(2)$	R(ReKi)	horizontal vector from furling hinge (frame-of-

		reference origin) to hub center (frame-of-reference origin)
<i>rTowerBaseHub(2)</i>	R(ReKi)	Horizontal vector from tower base center (frame-of-reference origin) to hub center (frame-of-reference origin)
<i>tmpVector(3)</i>	R(ReKi)	Tmp vector to help in certain slick calcs
<i>VelocityVec(3)</i>	R(ReKi)	Wind velocity in global reference frame at the element of interest
<i>Iblade</i>	I	Counter for blade
<i>IElement</i>	I	Counter for blade element

**STEPS:**

1. Check that aerodyn has been initialized (*Initialized*) else warn user and RETURN with ErrStat=1
2. If *NoLoadsCalculated* (parent module *Aerodyn*) OR the difference between the scaled CurrentTime\*OnePlusEpsilon and OldTime (*AeroTime*) is greater or equal to DTAERO (*AeroTime*) THEN:
  - a. It is time to update the aero forces
  - b. Set *DT*=*CurrentTime*(input)-*OldTime*
  - c. Set *OldTime*=*CurrentTime* (advances time stamp)
3. ELSE:
  - a. Just return current loads
  - b. IF NOT(*CurrentADOptions*%*LinearizeFlag*) (input) THEN
    - i. *AD\_CalculateLoads* = *ADCcurrentLoads* (parent module *Aerodyn*)
    - ii. RETURN
4. Set *Time* = *CurrentTime*
5. Calculate Rotor Speed *REVS* (**Rotor**) as DOT\_PRODUCT of the direction of the hub rotational axis (x-axis) and the difference between the rotational velocity of the Hub and the furling rotational velocity (from  $\underline{\omega}_{ROT} = \underline{\omega}_{Furl} + \underline{\omega}^{rel} = \underline{\omega}_{Furl} + \begin{bmatrix} REVS \\ \underline{\omega}^{rel} \cdot \hat{j} \\ \underline{\omega}^{rel} \cdot \hat{k} \end{bmatrix}$ ) where  $\underline{\omega}^{rel}$  is the rotational velocity relative to the reference frame fixed with the furling system
6. Calculate *YawAngle* (**Rotor**) as -ATAN2 of the RotorFurl direction cosine components (1,2) and (1,1), then calculate sin and cos of the *YawAngle*

7. Calculate *Tilt (Rotor)* as ATAN2 of the RotorFurl direction cosine components (1,3) and the norm of the direction cosine vector [(1,1) (1,3)]; then calculate sin and cos of *Tilt*
8. Calculate some yaw rotational velocity components (local variables):
  - a. *AvgVelNacelleRotorFurlYaw* as difference between the #3 components of the RotorFurl rotational velocity and Nacelle rotational velocity; this represents the relative rotational velocity of the rotor w.r.t. nacelle (z component) ( $\underline{\omega}_F^{rel2N}$ )
  - b. *AvgVelTowerBaseNacelleYaw* as difference between the #3 components of the Nacelle rotational velocity and Tower rotational velocity; this represents the relative rotational velocity of the nacelle w.r.t. tower (z component) ( $\underline{\omega}_N^{rel2T}$ )
  - c. *AvgVelTowerBaseYaw* as the #3 component of the Tower rotational velocity ( $\underline{\omega}_T$ )
  - d. *rRotorFurlHub(1:2)* as the horizontal vector from furling hinge (frame-of-reference origin) to hub center (frame-of-reference origin) ( $\overrightarrow{FH}$ )
  - e. *rNacelleHub(1:2)* as the horizontal vector from Nacelle center (frame-of-reference origin) to hub center (frame-of-reference origin) ( $\overrightarrow{NH}$ )
  - f. *rTowerBaseHub(1:2)* as the horizontal vector from Tower Base center (frame-of-reference origin) to hub center (frame-of-reference origin) ( $\overrightarrow{TH}$ )
9. Calculate *YawVel* as a velocity directed along a horizontal direction on the rotor plane (y-hub axis see *DiskVel (AeroSubs)*) due solely to rotational velocities around the vertical global z axis of tower, nacelle and furl system. This is a bit unclear as the other contributions are neglected, not sure what this is for then. It should be:

$$\underline{v}_H = \underline{v}_T + \underline{v}_N^{rel2T} + \underline{v}_F^{rel2N} + \underline{\omega}_T \times \overrightarrow{TH} + \underline{\omega}_N^{rel2T} \times \overrightarrow{NH} + \underline{\omega}_F^{rel2N} \times \overrightarrow{FH}$$

Where H=hub, T=TwrBase, N=Nacelle, F=FurlSystem. The third term may be zero for a rigid nacelle. The routine calculates only the contribution due to the last three terms and disregards the components along X and Y of the  $\underline{\omega}$ 's. Those would be due to structural deflections of the various turbine components.

10. Calculate *AvgInfl (Rotor)* as  $2*SumInfl / (NB * R^2)$ .
11. Reset *SumInfl*=0.0
12. Call *DiskVelQ (AeroSubs)* which calculates the rotor-disk mean velocity and then it sets certain variables relative to *Wind*
13. If *DSTALL (Switch)* then Call *BedUpdate() (AeroSubs)*
14. If *Wake (Switch)* THEN Call *Inflow() (AeroSubs)*
15. CYCLE on all blades *IBlade*=1,NB (*Rotor*):
  - a. Calculate the azimuth angle *AzimuthAngle* for each blade simply through a slick way to use the ATAN2 and the DOT\_PRODUCT of the normals  $\hat{j}, \hat{k}$  respectively of the hub and furling systems; i.e. use *TurbineComponents%Hub%Orientation(3,:)* and *TurbineComponents%RotorFurl%Orientation(2:3,:)* (TYPE *AeroConfig* in *SharedTypes*); also account for spacing between blades equal to  $2*\pi/NB$
  - b. CYCLE on all blade elements *IElement*=1,*Nelm* (*Element*):

- i. Calculate *PitNow* (**Element**), i.e. the element pitch through a slick DOT\_PRODUCT of the local element and blade vectors  $\hat{i}$  and  $\hat{j}$ ; i.e. use *TurbineComponents%Blade(IBlade)%Orientation(1,:)* and *InputMarkers%Blade(IElement,IBlade)%Orientation(1:2,:)* (TYPE **AllAeroMarkers** in **SharedTypes**); Note this is a pitch calculated w.r.t. a plane normal to the pitch axis, not the local plane of rotation (if there is coning for instance)
- ii. Calculate *SPitch* and *CPitch* (internal variables)
- iii. Calculate *tmpVector* as distance vector between Hub and Blade Element(*IElement,IBlade*) (using *InputMarkers* and *TurbineComponents*)
- iv. Calculate *rLocal* (internal variable) as the norm of the component of *tmpVector* in the (y,z) plane of the Hub, i.e. the plane of rotation
- v. If the input *CurrentADOptions%SetMulTabLoc(IElement,IBlade)* is .TRUE. THEN set *MulTabLoc (Airfoil)* = *CurrentADOptions%MulTabLoc(IElement,IBlade)*
- vi. Calculate wind velocity at the blade element accounting for tower effects *VelocityVec(:)* = *AD\_WindVelocityWithDisturbance(InputMarkers%Blade(IElement,IBlade)%Position(:))* (AeroSubs)
- vii. Calculate the square of the wind velocity normal to the rotor plane: *VelNormalToRotor2* = ( *VelocityVec(3) \* STilt + (VelocityVec(1) \* CYaw - VelocityVec(2) \* SYaw) \* CTilt* )\*\*2; WHY NOT *VNWWind\*\*2*?
- viii. Construct the following local vectors:
- ix. *tmpVector*: this helps in the calculation of *VTTot* and *VNWWind* and *VNElement*. It uses a trick where rotation matrix components are multiplied by sin and cos of pitch angle and then summed so that the following vector can be calculated:
- x. *VTTot*, component of *VelocityVec(:)* - *TranslationVel(:)* (due to deflections and body motions) in the plane normal to the blade pitch axis and along the tangent to the rotation cone, following the convention for the local reference frame for the blade element. Rem: y along chord pointing towards the TE, z towards blade tip.
- xi. *VNWWind*, component of *VelocityVec(:)* normal to the tangent to the rotation circle and the blade pitch axis, not necessarily normal to the plane of rotation if coning or tilt exist following the convention for the local reference frame for the blade element.
- xii. *VNElement*, component of *TranslationVel (:)* normal to the tangent to the rotation circle and the blade pitch axis, not necessarily normal to the plane of rotation if coning or tilt exist following the convention for the local reference frame for the blade element.
- xiii. NOW Get blade element forces and induced velocity : Call *ELEMFRC( AzimuthAngle, rLocal, IElement, IBlade, VelNormalToRotor2, VTTot, VNWWind, VNElement, DFN, DFT, PMA, NoLoadsCalculated )* (AeroSubs)
- xiv. IF *DynInfl* .OR. *DynInit* (**Switch**) THEN: Setup dynamic inflow params

1. Call *GetRM((rLocal, DFN, DFT, AzimuthAngle, IElement, IBlade)* (*AeroSubs*)
- xv. HERE ARE THE LOADS: Recombine forces along the airfoil (x,y,z) as per unit length (divide by the DR):
  1. *ADCCurrentLoads%Blade(IElement,IBlade)%Force(1)* (parent module *Aerodyn*) = ( *DFN\*CPitch + DFT\*SPitch* ) / *DR(IElement)* (*Blade*)
  2. *ADCCurrentLoads%Blade(IElement,IBlade)%Force(2)* = ( *DFN\*SPitch - DFT\*CPitch* ) / *DR(IElement)*
  3. *ADCCurrentLoads%Blade(IElement,IBlade)%Force(3)* = 0.0
  4. *ADCCurrentLoads%Blade(IElement,IBlade)%Moment(1)* = 0.0
  5. *ADCCurrentLoads%Blade(IElement,IBlade)%Moment(1)* = 0.0
  6. *ADCCurrentLoads%Blade(IElement,IBlade)%Moment(3)* = *PMA* / *DR(IElement)*
- xvi. IF user requested to output wind components at the current element store them:  
IF(*WndElPrList(IElement)> 0 (ElOutParams)*) THEN:
  1. *SaveVX( WndElPrList(IElement), IBlade ) = VelocityVec(1)*
  2. *SaveVY( WndElPrList(IElement), IBlade ) = VelocityVec(2)*
  3. *SaveVZ( WndElPrList(IElement), IBlade ) = VelocityVec(3)*
- xvii. END\_LOOP on elements
- c. For blade #1 save local wind velocity for the last blade element if requested: IF (*IBlade==1 .AND. ElmPrn*) THEN:
  - i. *VXSAV = VelocityVec(1) , VYSAV = VelocityVec(2) , VZSAV = VelocityVec(3)*
- d. END\_LOOP on blades
16. SET *NoLoadsCalculated* (parent module *Aerodyn*) =.FALSE.
17. Set output *AD\_CalculateLoads* = *ADCCurrentLoads*

## OUTPUT

*AD\_CalculateLoads*: TYPE(**AllAeroLoads**) (*SharedTypes*)

This function returns the aerodynamic loads calculated at the blade element locations (the input marker locations).

## APPENDIX C. AERO<sub>SUBS</sub> MODULE

### GENERAL ORGANIZATION

AeroSubs[f90] contains the module *AeroSubs* which is the main set of subroutines and functions for AeroDyn.

### EXTERNAL (INVOKED) MODULES

- *NWTC\_LIBRARY* [see NWTC\_Library.f90]
- *InflowWind* [see InflowWindMod.f90]
- *SharedInflowDefns* [see SharedinflowDefs.f90]

### SUBROUTINES:

Name	Description
<u><i>ABPRECOR</i></u>	Adams-Bashforth Predictor/Corrector Integrator
<u><i>AD_GetInput</i></u>	It gets the main input for AeroDyn
<u><i>ADOOut</i></u>	It writes the ".opt" file
<u><i>AeroDyn_Terminate</i></u>	It terminates the program
<u><i>ATTACH</i></u>	1 <sup>st</sup> step in the LBM
<u><i>AXIND</i></u>	It calculates the induction factors
<u><i>BEDDAT</i></u>	It sets Beddoes parameters
<u><i>BEDINIT</i></u>	It calculates at 0-th time step many Beddoes parameters, including separation f's
<u><i>BEDDOES</i></u>	LBM driver
<u><i>BedUpdate</i></u>	It updates Beddoes parameters
<u><i>BEDWRT</i></u>	It writes Beddoes parameters to ".opt" file
<u><i>CheckRComp</i></u>	It performs checks on the input radial stations
<u><i>CLCD</i></u>	It interpolates Cl Cd Cm from airfoil data file based on AOA
<u><i>DynDebug</i></u>	It writes out information useful for debugging
<u><i>DiskVel</i></u>	It calculates the mean velocities relative to the rotor disk
<u><i>ELEMFRC</i></u>	It calculates loads on Blade Element
<u><i>GAUSSJ</i></u>	It inverts a square matrix via Gauss-Jordan's Method
<u><i>GetPhiLq</i></u>	It calculates value of integrals of BE forces times radial shape functions,to calculate $\tau_n^{mc}$ and $\tau_n^{ms}$

<u>GetTipLoss</u>	It computes Tip-Loss for blade element
<u>GetPrandtlLoss</u>	It computes generic Prandtl-Loss for blade element
<u>GetRM</u>	Subroutine that returns the mode-moments of the blade elemental force
<u>GetTwrInfluence</u>	It computes tower shadow or dam influence on the blade
<u>GetTwrSectProp</u>	It returns the tower radius and Cd for the local vertical tower station
<u>InfDist</u>	It calculates the states of GDW at the current time-step
<u>Infinit</u>	It initializes GDW and calculates some matrices that are constant throughout
<u>Inflow</u>	Subroutine that leads to the GDW routines
<u>infupd</u>	It updates GDW states
<u>LMATRIX</u>	It calculates the L_cos and L_sin matrices,i.e. $[\tilde{L}^c]$ and $[\tilde{L}^s]$
<u>MATINV</u>	It inverts the [L_cos] and [L_sin] matrices
<u>READFL</u>	It reads in Airfoil File Data
<u>READTwr</u>	It reads in Tower File Data
<u>SEPAR</u>	2 <sup>nd</sup> step in the LBM To review just for the highlights
<u>VIND</u>	It calculates the axial induction factor
<u>VINDERR</u>	It sends some error messages to the screen
<u>VindInf</u>	It calculates $\alpha$ and $\alpha'$ from GDW results
<u>VNMOD</u>	It applies skewed wake correction (yaw/tilt)
<u>VORTEX</u>	The 3 <sup>rd</sup> step in the LBM
<u>WindAzimuthZero</u>	It sets 0-azimuth in a wind reference frame

**FUNCTIONS:**

Name	Description
<u>AD_WindVelocityWithDisturbance</u>	It computes the wind velocity components at the location of interest, including any tower effect
<u>FGAMMA</u>	It calculates the $\Gamma_{ln}^{km}$ function for the [L] matrices
<u>GetReynolds</u>	It calculates the blade-element (chord) Reynolds number

<u>HFUNC</u>	It calculates $H_n^m$ used in the [M] matrix and radial shape functions
<u>IDUBFACT</u>	Calculates the double factorial of an integer
<u>PHIS</u>	It calculates the $\varphi_l^k(v)$ from definition (see Theory Sections); Currently replaced by <u>XPHI</u>
<u>SAT</u>	Saturation trimming function
<u>XPHI</u>	It calculates the radial shape function values at the local radial distance r; only for 3 harmonics, and max power: $r^3$

## C1. READFL()

Subroutine that reads airfoil data file (Cl, Cd, Cm, and dynamic stall parameters). Called by [AD\\_GetInput](#) (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

- *Airfoil* [see AeroMods.f90]
- *AD\_IOParams* [see AeroMods.f90]
- *AeroGenSubs* [see GenSubs.f90]
  - Only subroutine *AllocArrays*
- *Switch* [see AeroMods.f90]
- *Bedoes* [see AeroMods.f90]

### INPUT

The input is basically given through Module variables, e.g.: *NUMFOIL* (*Airfoil*)

Name	Type/Module	From	Description
<i>UnAirfl</i>	I(4)/ <i>AD_IOParams</i>	<i>AeroMods</i> [.f90]	Airfoil file unit
<i>NUMFOIL</i>	I(4)/ <i>Airfoil</i>	<i>AeroMods</i> [.f90]	Number of airfoil files
<i>FOILNM</i> (:)	C(1024)/ <i>Airfoil</i>	<i>AeroMods</i> [.f90]	Names of airfoil files

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>CDNegPI</i> (:)	R(ReKi)			( <i>Ntables(IPHI</i> ) Cds at -pi
<i>CDPosPI</i> (:)	R(ReKi)			( <i>Ntables(IPHI</i> ) Cds at +pi
<i>CLNegPI</i> (:)	R(ReKi)			( <i>Ntables(IPHI</i> ) Cls at -pi
<i>CLPosPI</i> (:)	R(ReKi)			( <i>Ntables(IPHI</i> ) Cls at +pi
<i>CMNegPI</i> (:)	R(ReKi)			( <i>Ntables(IPHI</i> ) Cms at -pi
<i>CMPosPI</i> (:)	R(ReKi)			( <i>Ntables(IPHI</i> ) Cms at +pi
<i>IPHI</i>	I(4)			Counter for cycles on airfoil tables
<i>I</i>	I(4)			Counter
<i>K</i>	I(4)			Counter
<i>Sttus</i>	I(4)			Status Parameter
<i>NFOILID</i>	I(4)			Counter for airfoil files
<i>NumLines</i>	I(4)			Number of lines in airfoil file being read

<i>NUNIT</i>	I(4)	Unit for airfoil file
<i>IOS</i>	I	IO-status switch
<i>ALPosPI</i>	L	True if AOA=180 is present in the current airfoil file
<i>ALNegPI</i>	L	True if AOA=-180 is present in the current airfoil file
<i>TITLE(2)</i>	C(40)	Title and subtitle from airfoil files
<i>LINE</i>	C(1024)	1 line of the airfoil file read in

### STEPS

1. Set *NUNIT*=*UnAirfl (AD\_IOParams)* and *NumCL (Airfoil)*=0
2. In a DO\_LOOP, one cycle per airfoil file *NFOILID* = 1, *NUMFOIL*:
  - a. Open an airfoil file, reads the number of data lines past the header
  - b. Find the maximum number of lines (for only the data such as AOA or Cl) across all files: *NumCL (Airfoil)*
  - c. Close the file
3. Call *AllocArrays('Aerodata')* (*AeroGenSubs*) which allocates Airfoil Cl,Cd, etc. and dynamic stall arrays, based on *NumFoil* and *NumCL (Airfoil)*
4. In a DO-LOOP, one cycle per airfoil file *NFOILID* = 1, *NUMFOIL*:
  - a. Open an airfoil file
  - b. Read title and subtitle *TITLE(1:2)* (internal) and number of aifoil tables in the file *Ntables(:)* (*Airfoil*)
  - c. Allocate some local arrays (*CLPosPi..CMNegPi*) based on *Ntables*
  - d. Read the *MultTabMet(NFOILID,Ntables)* (module *Airfoil*), e.g. the Reynolds numbers in the file
  - e. Read 4 junk lines no longer used
  - f. Read Beddoes (*Beddoes*) stall parameters (*AOL, CNA, CNS, CNSL, AOD ,CDO* all *R(Ntables)*) and convert angles *AOL, AOD* to radians
  - g. Initialize *NLIFT(NFOILID)* =0 and *ALPosPI* = .FALSE. and *ALNegPI* = .FALSE.
  - h. Read till end of file and through all the *Ntables(NFOILID)* tables: *AL, CL, CD, CM* (*Airfoil*) with *CM* only if *PMOMENT(Switch)* was enabled, and do some checks to see whether data is reasonable, and store values (*CLPosPi..CMNegPi*) at *AL*=±180°.
  - i. Transform *AL(:, :)* into radians
  - j. Assign *NLIFT(:)* (module *Airfoil*), which is number of data points per file
  - k. Close File
  - l. Check that stored values at *AL*=±180° match, then deallocate local arrays no longer needed

### OUTPUT

The Airfoil data is filled: *Ntables()*, *NLIFT()*, *AL*, *CL*, *CD*, *CM*, *AOL*, *CNA*, *CNS*, *CNSL*, *AOD*, *CDO* (modules *Airfoil* and *Beddoes*)

## C2. AD\_GETINPUT (UNIN, AEROINFILE, WINDFILENAME, TITLE, ERRSTAT)

Subroutine that opens the overall input file for AeroDyn, reads in all the options to run the program, the airfoil aerodynamic data (polar curves) plus dynamic stall data; it also initializes parameters and variables needed for the LBM dynamic stall modeling. Called by *AD\_Init* (*Aerodyn*).

### EXTERNAL (INVOKED) MODULES

- *Airfoil* [see AeroMods.f90]
- *AD\_IOParams* [see AeroMods.f90]
- *AeroTime* [see AeroMods.f90]
- *Blade* [see AeroMods.f90]
- *Rotor* [see AeroMods.f90]
- *EIOutParams* [see AeroMods.f90]
- *Element* [see AeroMods.f90]
- *InducedVel* [see AeroMods.f90]
- *TwrProps* [see AeroMods.f90]
- *Wind* [see AeroMods.f90]
- *AeroGenSubs* [see GenSubs.f90] (only *AllocArrays*)
- *Switch*[see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
<i>UnIn</i>	I	IN	Unit number for aerodyn input file
<i>AeroInFile</i>	C(*)	IN	AeroDyn input file
<i>WindFileName</i>	C(*)	OUT	Wind file name
<i>TITLE</i>	C(*)	OUT	Used for <u><i>ADOUT()</i></u>
<i>ErrStat</i>	I	OUT	Returns 0 if no errors were encountered; non-zero otherwise

Additional Input is provided via module variables (e.g.: *TwrShadow* or *TwrPotent* module *TwrProps*)

### LOCAL VARIABLES

A few auxiliary variables needed as counters in DO\_LOOP, or string variables to read in data from files.

Name	Type	Value	PARAMETER/ Initialized	Description
<i>ElIndex</i>	I			Index of element-info array for printing
<i>IElm</i>	I			Counter
<i>IndPrint</i>	I			Temporary index within 1 line-string to find substrings in the input

K	I	Counter on airfoil files
Line C(1024)		1 line worth of input file

**STEPS**

1. It opens *AeroInfile* (Aerodyn Input File) unit *UnIn*
2. It checks whether the *NWTC\_IO/Library* Echo flag is activated and if so it writes a message on the Echo file
3. Read in the title and writes to screen the heading of the input file
4. Reads in unit system and makes sure units are SI
5. Reads in stall model, *PMoment* (pitching moment option),
6. Read in Inflow model: BEM vs Dynamic Wake; additionally 'DA','DB','DT' strings that decide whether or not to use drag multiplier *EqAIDmult* (*InducedVel*) (either 0 or 1) in induction factor calculation: 'DA' only in axial momentum, 'DT' only in tangential (torque) momentum, 'DB' in both.
7. Set accordingly the logical variables *EqAIDmult*, *EquilDA* (*Switch*), *EquilDT* (*Switch*), *DynInfl* (*Switch*), *DynInit* (*Switch*)
8. Read in wake model, tolerance for convergence criteria, Tip-loss model, Hub-loss model
9. Read in *WindFileName* (see also *InflowWindMod*), HH reference height
10. Read in either 'NEWTOWER'(string for new tower model) or old tower model *TwrShadow*:
  - a. IF 'NEWTOWER' model THEN:
    - i. set *PJM\_Version*=.TRUE.
    - ii. read *TwrPotent* (*TwrProps*)
    - iii. read *TwrShadow* (*TwrProps*)
    - iv. read *TwrFile* (*TwrProps*)
  - b. ELSE:
    - i. set *PJM\_Version*=.FALSE.,
    - ii. set *TwrPotent*=.FALSE.,
    - iii. set *TwrShadow*=.FALSE.
    - iv. read in Tower Shadow Deficit *TwrShad* (*TwrProps*) and checks it is a reasonable numeric number (<1), in case it is >1 warn user and set *TwrShad*=0.3
    - v. IF *Echo* is set to .TRUE. THEN write to *UnEc* unit of echo file (*NWTC\_Library*)
    - vi. Read in Shadow (Normalized by Tower local Radius) HalfWidth: *ShadHWid* (*TwrProps*);
    - vii. IF *ShadHWid* <0.0 THEN warn user and set *ShadHWid* =1.0
    - viii. Read in tower Shadow Reference Point *T\_Shad\_Refpt* (*TwrProps*), distance from tower center to hub
    - ix. Set constants used in tower shadow calculations:
      1.  $TShadC1 = ShadHWid / \text{SQRT}(\text{ABS}(T_Shad_Refpt))$  (*TwrProps*)
      2.  $TShadC2 = TwrShad * \text{SQRT}(\text{ABS}(T_Shad_Refpt))$  (*TwrProps*)
11. Read in air density and kinematic viscosity *Rho* and *KinVisc* (*Wind*)

12. Checks if it is ok to proceed with GDW: *Rho* must be  $>0$
13. Read in *DtAero* (*AeroTime*), aero-calculation time step, and number of airfoil files *NumFoil* (*Airfoil*) making sure it is  $>0$  else warn user and RETURN
14. Allocate the array containing the names of the airfoil data files *FoilNm*( *NumFoil* ) (*Airfoil*)
15. Read airfoil data file names from the next *NumFoil* lines in the input file: *ReadAryLines*( *UnIn*, *AeroInFile*, *FoilNm*(1:*NumFoil*), *NumFoil*, 'FoilNm', 'Airfoil file names', *ErrStat* ) (*NWTC\_IO* within *NWTC\_Library* package)
16. Read in number of blade elements *NElm* (*Element*)
17. Call *AllocArrays*('Element') (*AeroGenSubs*) to allocate space for blade element data
18. Initialize *NumElOut* =0 and *NumWndElOut*=0 (*ElOutParams*)
19. DO\_LOOP cycle *Ielm*=1:*NElm* to read all the element information in the file
  - a. Read headings and then the following file lines with element information and whether or not to print element output, and whether or not to print Wind data for that element:
    - i. *RElm*(*Ielm*) and *Twist*(*Ielm*) (*Element*) *DR*(*Ielm*) and *C*(*Ielm*) (*Blade*) *NFoil*(*Ielm*) (*Airfoil*)
    - ii. IF string "PRINT" is encountered update *ElPrList*(*Ielm*) *NumElOut* (*ElOutParams*)
    - iii. IF string "WIND" is encountered update *WndElPrList*(*Ielm*) *NumWndelOut* (*ElOutParams*)
  - b. Check whether Echo file (*NWTC\_IO* within *NWTC\_Library*) was activated in *NWTC\_IO.f90* (debug purposes) and write to it in case, and do some more checks that all is going with no errors, that chords>0 and that ID numbers are between 1 and *NumFoil*
  - c. Convert *Twist*(:) from deg to radians
20. Read in Multiple Table option (which may or may not be there, **not sure about this one**), IF not present THEN set:
  - a. *MultiTab*=.FALSE. (*Switch*)
  - b. *Reynolds*=.FALSE. (*Switch*)
21. Close Input File *UnIn*
22. Read all airfoil files's data (Call *READFL* in *Aerosubs*)
23. Initialize *MulTabLoc*=0.0
24. IF *NTables*(:) showed any value greater than 1 but no multi-table option was found in the input file THEN warn user
25. ELSEIF *NTables*(:) showed any value greater than 1 (also multi-table option was found) THEN, based on multi-table option:
  - a. 'USER': set *MultiTab* = .TRUE. and *Reynolds* = .FALSE.
    - i. Cycle on *NumFoil* to check whether ranges of *MulTabMet*(*NFOILID*,*Ntables*(*NFOILID*)) are compatible, **this is very strange**
  - b. 'RENUM' : set *MultiTab* = .TRUE. and *Reynolds* = .TRUE.
  - c. 'SINGLE': set *MultiTab* = .FALSE. and *Reynolds* = .FALSE.
  - d. Everything else warn user

26. ELSE set *MultiTab* = .FALSE. and *Reynolds* = .FALSE.
27. IF *TwrPotent* .OR. *TwrShadow* (*TwrProps*) THEN Call *READTwr*(*UnIn*, *TwrFile*, *ErrStat*) to read in tower file (*AeroSubs*)
28. If *NumElOut* > 0. .OR. *NumWndElOut*>0 THEN
  - a. Set *ElemPrn* = .TRUE. (*Switch*)
  - b. Allocate arrays for printing element information if requested: call *AllocArrays*('ElPrint') (*AeroGenSubs*)
  - c. Re-initialize *WndElPrNum*(:) and *ElPrNum*(:) (*ElOutParams*)
29. If *Dstall* call *BEDDAT* (*AeroSubs*) to initialize Bedoes parameters

## OUTPUT

It stores in the appropriate variables (defined under the invoked modules) airfoil aerodynamic data, Bedoes's initial parameters and Kirchoff's 'f' tables for each airfoil and airfoil table values for both CN and CC. It also stores in the appropriate variables all the options to run the program, and further variables for the printing of element information.

### C3. READTWR (UNIN, FILENAME, ERRSTAT)

Subroutine that reads the tower properties input file, allocating *TwrProps* variables to do so. The tower data file contains radius and Re vs CD data as well as the tower wake constant. Called by *AD\_GetInput* (*AeroSubs*).

#### EXTERNAL (INVOKED) MODULES

- *TwrProps* [see AeroMods.f90]

#### INPUT

Name	Type	INTENT	Description
<i>UnIn</i>	I	IN	Unit number for tower input file
<i>FileName</i>	C(*)	IN	Name of the tower input file
<i>ErrStat</i>	I	OUT	Returns 0 if no errors were encountered; non-zero otherwise

#### LOCAL VARIABLES

A few auxiliary variables needed as counters in DO\_LOOP, or format strings.

Name	Type	Value	PARAMETER/ Initialized	Description
<i>I</i>	I			Loop counter for rows in the data tables
<i>J</i>	I			Loop counter for columns in the data tables
<i>Fmt</i>	C(99)			Format for printing to an echo file

#### STEPS

1. Open *FileName*, tower file and read Title's line 1 and line 2
2. Read in *NTwrHt* number of tower height stations, *NTwrRe* number of tower Re values, *NTwrCD* number of tower CD columns, *Tower\_wake\_constant* (all from *TwrProps* module)
3. Allocate *TwrHtFr(NTwrHt)*, *TwrWid(NTwrHt)*, *NTwrCDCol(NTwrHt)*, *TwrRe(NTwrRe)*, *TwrCD(NTwrRe, NTwrCD)* (all from *TwrProps* module)
4. Skip following headers and read in *TwrHtFr(1:NTwrHt:)*, *TwrWid(1:NTwrHt)*, *NTwrCDCol(1:NTwrHt)*
5. Do some reasonability checks on the values
6. Skip the next two header lines and read in *TwrRe(1:NTwrRe)* and *TwrCD(1:NTwrRe,1:NTwrCD)*, i.e. various Cds associated with each Reynolds number. In fact various stations may be associated with different Cd at the same Reynolds number.
7. Write to Echo similar info if *Echo(NWTC\_IO)* activated.
8. Close *FileName*

#### OUTPUT

*Errstat*: I; usual error status flag.

It reads in the Tower drag file, and stores properties into variables associated with *TwrProps* module.

## C4. CHECKRCOMP(ADFILE, HUBRADIUS, TIPRADIUS, ERRSTAT)

Subroutine that checks whether  $RElm(:)$  and  $DR(:)$  are compatible within a millimeter. Called by *AD Init* (*Aerodyn*).

### EXTERNAL (INVOKED) MODULES

- *Blade* [see AeroMods.f90]
  - Only  $DR$
- *Element* [see AeroMods.f90]
  - Only  $NElm, RELm$

### INPUT

Name	Type	INTENT	Description
<i>ADFile</i>	C(*)	IN	Name of aerodyn input file
<i>HubRadius</i>	R(ReKi)	IN	Hub Radius
<i>TipRadius</i>	R(ReKi)		Tip Radius
<i>ErrStat</i>	I	OUT	Returns 0 if no errors were encountered; non-zero otherwise

Additional input via variables in the invoked modules.

### LOCAL VARIABLES

A few auxiliary variables needed as counters in DO\_LOOP, or format strings.

Name	Type	Value	PARAMETER/ Initialized	Description
<i>DRNodesNew(NElm)</i>	R(ReKi)			Lengths of variable-spaced blade elements
<i>DRSum</i>	R(ReKi)			Sum of DRs
<i>EPS</i>	R(ReKi)	$EPSILON(HubRadius)$	PARAMETER	A small value used to compare two numbers
<i>DRChange</i>	C(33)			A string showing how to change DR to get compatibility

### STEPS

1. Initialize  $ErrStat=0$
2. Calculate  $DRNodesNew(1)$  as  $(RElm(1) - HubRadius)^2$  and initialize  $DRSum = DRNodesNew(1) + HubRadius$
3. Check that  $DRNodesNew(1)$  is within 1 mm of  $DR(1)$  AND that  $RElm(1) > HubRadius$  else cause program to abort

4. Calculate  $DRNodesNew(2:Nelm)$  as  $2.0 * (Relm(i) - Relm(i-1)) - DRNodesNew(i-1)$  with  $i=2:Nelm$  (not exactly clear how this DR should work, it seems inconsistent, with every other DR smaller than the following one); update  $DRSum$  by summing the new  $DRNodesNew(i)$
5. Check that  $DRNodesNew$  is greater than  $EPS$  and that it is within 1 mm of user inputted  $DR$ , if not cause program to abort
6. Check that  $DRSum$  is within 1 mm of  $TipRadius$ , else cause program to abort

## OUTPUT

It does not have a real output, it prints warning and error messages on screen if input blade element radii and DRs are not supposed to be correct and may abort program, so  $ErrStat$  is the only output of the check.

## C5. ADOUT (TITLE, HUBRAD, WINDFILENAME)

Subroutine that writes the “.opt” file (a sort of input echo file). Called by *AD Init* (*AeroDyn.f90*).

### EXTERNAL (INVOKED) MODULES

- *AD\_IOParams* [see *AeroMods.f90*]
- *EIOutParams* [see *AeroMods.f90*]
- *AeroTime* [see *AeroMods.f90*]
- *Airfoil* [see *AeroMods.f90*]
- *Blade* [see *AeroMods.f90*]
- *Element* [see *AeroMods.f90*]
- *InducedVel* [see *AeroMods.f90*]
- *Rotor* [see *AeroMods.f90*]
- *Switch* [see *AeroMods.f90*]
- *Wind* [see *AeroMods.f90*]
- *TwrProps* [see *AeroMods.f90*]

### INPUT

Name	Type	INTENT	Description
<i>TITLE</i>	C(*)	IN	Title to be written onto the .opt file
<i>HubRad</i>	R(ReKi)	IN	Hub radius to be written onto the .opt file
<i>WindFileName</i>	C(*)	IN	Wind file name to be written onto the .opt file

Additional Input is provided via module variables (e.g.: *TwrShadow* or *TwrPotent* module *TwrProps*)

### LOCAL VARIABLES

A few auxiliary variables needed as counters in DO\_LOOP, or string variables to prepare output strings for units in the headings of the opt file.

Name	Type	Value	PARAMETER/ Initialized	Description
<i>IElm</i>	I(4)			Counter for elements
<i>IFoil</i>	I(4)			Counter for airfoil files
<i>Dst_Unit</i>	C(2)			A small value used to compare two numbers
<i>Frmt</i>	C(150)			Format String
<i>Mass_Unit</i>	C(4)			Unit of Mass string
<i>MESAGE</i>	C(35)			Message string
<i>Vel_Unit</i>	C(3)			Unit of velocity string

Delim C(1)

Delimiter string

**STEPS**

1. It sets the unit strings according to SI vs. English
2. It writes on *UnADOpt (AD\_IOParams)* messages and echos the input file accordingly to selected options in *Switch* module variables and read in variables (such as tower properties and file names)
3. It writes on *UnADOpt (AD\_IOParams)* the Element information as read from the input file
4. It writes on *UnADOpt (AD\_IOParams)* Rotor Radius *R*, Hub Radius *HubRad* and Number of Blades *NB* (note: these are passed to the program, usually by *AD Init* which calculated them)
5. If *DSTALL(Switch)* then call *BEDWRT*
6. Close unit *UnADOpt* opt file

**OUTPUT**

It just writes to the opt file the input variables selected by user and a few more information

## C6. BEDDAT()

Subroutine that sets certain parameters used by the Dynamic Stall Routine; Called by *AD\_GetInput* (*AeroSubs*). In particular, it sets the initial values of Cn/Cna and Cc/Cna used for calculating the Kirchoff's f values.

### EXTERNAL (INVOKED) MODULES

- *Airfoil* [see AeroMods.f90]
- *Beddoes* [see AeroMods.f90]
- *Switch* [see AeroMods.f90]

### INPUT

Input is via variables within invoked modules, *Airfoil* in particular for *CL*, *CD*, *AL,CNA,CDO*, *NLift*, *NTables*, *NUMFOIL*.

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
ETA	R(ReKi)	.99	Initialized	This should be the recovery factor, but it does not get used
CA	R(ReKi)			Cos of AOA read from airfoil file
SA	R(ReKi)			Sin of AOA read from airfoil file
I	I(4)			Index for AOA line within airfoil file
J	I(4)			Index for airfoil file
K	I(4)			Index for <i>Ntables</i> within airfoil data file

### PARAMETERS SET AND VARIABLES INITIALIZED IN THIS ROUTINE (BEDDOES MODEL CONSTANTS)

Name	Type/ Beddoes Module	Value	PARAMETER/ Initialized	Description
TVL	R(ReKi)	11.0	Initialized	Non-dimensional transit time constant for vortex moving along airfoil
TP	R(ReKi)	1.7	Initialized	Pressure lag time constant
TV	R(ReKi)	6.0	Initialized	Time constant for strength of shed vortex
TF	R(ReKi)	3.0	Initialized	Time constant applied to location of the separation point
AS	R(ReKi)	335	Initialized	Speed of Sound (m/s)
VOR	L	.FALSE.	Initialized	Flag that checks whether or not LE vorticity should be

					increased or decay		
SHIFT	L	.FALSE.	Initialized	Flag that checks whether or not AOA is increasing (toward stall)			
BEDSEP	L	.FALSE.	Initialized	Flag set when LE vortex is allowed to advect			
ANE1	R(ReKi)	0.0	Initialized	Previous time step value of ANE, effective AOA at $\frac{3}{4}$ -chord			
OLDCNVR(ReKi)	0.0	Initialized	Previous time step value of CNV ( <i>Bedoes</i> )				
CVN1	R(ReKi)	0.0	Initialized	Previous time step value of CVN ( <i>Bedoes</i> )			
CNPOT1	R(ReKi)	0.0	Initialized	Previous time step value of CNPOT ( <i>Bedoes</i> )			
CNP1	R(ReKi)	0.0	Initialized	Previous time step value of CNP ( <i>Bedoes</i> )			
CNPD1	R(ReKi)	0.0	Initialized	Previous time step value of CNPD ( <i>Bedoes</i> )			
OLDDF	R(ReKi)	0.0	Initialized	Previous time step value of DF ( <i>Bedoes</i> )			
OLDDFCR(ReKi)	0.0	Initialized	Previous time step value of DFC ( <i>Bedoes</i> )				
OLDDPPR(ReKi)	0.0	Initialized	Previous time step value of DPP ( <i>Bedoes</i> )				
FSP1	R(ReKi)	0.0	Initialized	Previous time step value of FSP ( <i>Bedoes</i> )			
FSPC1	R(ReKi)	0.0	Initialized	Previous time step value of FSPC ( <i>Bedoes</i> )			
TAU	R(ReKi)	0.0	Initialized	Non-dimensional time to track advection of LE vortex			
OLDTAUR(ReKi)	0.0	Initialized	Previous time step TAU				
OLDXN	R(ReKi)	0.0	Initialized	Previous time step value of XN ( <i>Bedoes</i> )			
OLDYN	R(ReKi)	0.0	Initialized	Previous time step value of YN ( <i>Bedoes</i> )			

## STEPS

1. Initialize main parameters (see above table): TVL through OLDYN
2. DO\_LOOP  $J=1, NUMFOIL$  : i.e. on every airfoil file
  - a. DO\_LOOP  $K=1, Ntables(J)$ ; i.e. on every table within the airfoil file
    - i. DO\_LOOP  $I=1, NLIFT(J)$ : i.e. on every AOA line within the airfoil table
      1. Calculate cos and sin of  $AL(J,I)$ , (*Airfoil*) AOA
      2. Calculate the Cn and Cc coefficients (normal force and chord force):
        - a.  $CN = CL(J,I,K) * CA + ( CD(J,I,K) - CDO(J,K) ) * SA$
        - b.  $CC = CL(J,I,K) * SA - ( CD(J,I,K) - CDO(J,K) ) * CA$ ; this is along the chord pointing towards the LE

3. Calculate  $FTB$  and  $FTBC$ , i.e. tabulation of  $Cn/Cna$  and  $Cc/Cna$  to be used later to calculate the actual  $f$  for the blade element under the actual AOA conditions:
  - a. IF  $CNA(J,K) > 1E-6$  THEN:
    - i.  $FTB(J,I,K) = CN / CNA(J,K)$
    - ii.  $FTBC(J,I,K) = CC / CNA(J,K)$
  - b. ELSE set  $FTB$  and  $FTBC(J,I,K)=1.0$

## OUTPUT

$FTB(J,I,K)$  and  $FTBC(J,I,K)$ : R(ReKi); *Beddoes* initial tables of  $Cn/Cna$  and  $Cc/Cna$  for the calculations of Kirchoff's 'f' values at a later time (see *BedInit (AeroSubs)*).

It also sets all the parameters for Beddoes (which perhaps could be set in modules like *AeroMods*[.f90]).

## C7. BEDINIT(J,IBLADE,ALPHA)

Subroutine that calculates initial values of Bedoes 'f' arrays; Called by ELEMFR (**AeroSubs**) at the first time step only.External (Invoked) Modules

- *Airfoil* [see AeroMods.f90]
- *Bedoes* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
<i>J</i>	I	IN	Blade element index
<i>IBlade</i>	I	IN	Blade index
<i>ALPHA</i>	R(ReKi)	INOUT	AOA

Many additional input via variables in the invoked modules. E.g.: *ANE* (**Bedoes**).

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>AOL1</i>	R(ReKi)			AOA at 0lift interpolated across airfoil tables
<i>CNA1</i>	R(ReKi)			Cn slope coefficient interpolated across airfoil tables
<i>FSPA</i>	R(ReKi)			Intermediate interpolation value for <i>FSP</i> ( <b>Bedoes</b> AeroMods.f90) along AOA
<i>FSPB</i>	R(ReKi)			Intermediate interpolation value for <i>FSP</i> ( <b>Bedoes</b> AeroMods.f90) along Airfoil Tables
<i>FSPCA</i>	R(ReKi)			As <i>FSPA</i> for <i>FSPC</i>
<i>FSPCB</i>	R(ReKi)			As <i>FSPB</i> for <i>FSPC</i>
<i>P</i>	R(ReKi)			Facto expression for linear interpolation
<i>P1</i>	R(ReKi)			Interpolation factor accounting for AOA deltas
<i>P2</i>	R(ReKi)			Interpolation factor accounting for airfoil Table parameter (e.g., Reynolds) deltas
<i>SRFP</i>	R(ReKi)			Signed, square root of f'
<i>TEMP</i>	R(ReKi)			Temporary variable used in the calculation of <i>f</i>
<i>I</i>	I(4)			Index for the airfoil file data to be used for the current <i>J</i> blade element

<i>I1</i>	<i>I(4)</i>	Interpolation lower index for base array to be interpolated
<i>I1P1</i>	<i>I(4)</i>	<i>I1+1</i>
<i>I2</i>	<i>I(4)</i>	Interpolation lower index for array to be interpolated
<i>I2P1</i>	<i>I(4)</i>	<i>I2+1</i>
<i>N</i>	<i>I(4)</i>	Index for <i>MulTabLoc</i> within <i>MulTabMet</i> array ( <i>Airfoil</i> )
<i>NP1</i>	<i>I(4)</i>	<i>N+1</i>

## STEPS

1. Set  $ANE(J,IBlade)$  (*Bedoes*)= *ALPHA*
2. Set  $AFE(J,IBlade)$  (*Bedoes*)= *ALPHA*
3. Set  $I = NFOIL(J)$  (*Airfoil*), index of airfoil file to use for the  $J$  element under consideration
4. IF multiple tables for the current airfoil:  $NTables(I)>1$  (*Airfoil*) THEN interpolate across tables for *CNA1* and *AOL1* airfoil constants:
  - a. Set  $MulTabLoc$  (*Airfoil*)=  $\text{MIN}(\text{MAX}(MulTabLoc, MulTabMet}(I,1) \text{ } (Airfoil)), MulTabMet(I,NTables(I))$  so that it stays within *MulTabMet*( $I,:)$  (e.g.:Reynolds) number bounds for instance; i.e. set the Table Parameter (*ElemFrc* (*AeroSubs*) might have set it to the Reynolds Number for instance)
  - b. Call *Locatebin* (*NWTC\_Num* within *NWTC\_Library*) to find the index  $N$  for the value just below the *MulTabLoc* value within the array *MulTabMet*( $I,1:NTables(I)$ ) (Note: this and other airfoil data parameters were read in via *AD\_GetInput/ReadFL* (*AeroSubs*))
  - c. IF  $N==[0/NTables(I)]$  THEN:
    - i. Set  $CNA1 = CNA(I,[1/N])$  (*Bedoes*) (Note: this and other airfoil data parameters were read in via *AD\_GetInput/ReadFL* (*AeroSubs*))
    - ii. Set  $AOL1 = AOL(I, [1/N])$  (*Bedoes*)
  - d. ELSE: regular interpolation to be performed:
    - i. Set  $NP1=N+1$
    - ii. Set  $P=(MulTabLoc-MulTabMet}(I,N))/(MulTabMet}(I,NTables(I))-MulTabMet}(I,N))$
    - iii.  $CNA1 = CNA(I,N) + P * (CNA(I,NTables(I))-CNA(I,N))$
    - iv.  $AOL1 = AOL(I,N) + P * (AOL(I,NTables(I))-AOL(I,N))$
5. ELSE: no interpolation needed, set:
  - a.  $CNA1 = CNA(I,1)$
  - b.  $AOL1 = AOL(I,1)$
6. Start composing the potential  $C_n$ :
  - a.  $CNPOT(J,IBLADE) = CNA1 * (\text{ALPHA} - AOL1)$
  - b.  $CNP(J,IBLADE) = CNPOT(J,IBLADE)$
7. Determine the indices for AOA, for an interpolation based on AOA, and the linear interpolation factor: (note: this could be more efficiently written, minor bug)

- a. Set  $\text{ALPHA} = \text{MIN}(\text{MAX}(\text{ALPHA}, \text{AL}(I,1) \text{ (Airfoil)}), \text{AL}(I,\text{NLIFT}(I) \text{ (Airfoil)}) )$  to keep it within the  $\text{AL}(I,:)$  range
  - b. Call Locatebin to find the index  $I1$  for the value just below the  $\text{ALPHA}$  value within the array  $\text{AL}(I,\text{NLIFT}(I))$
  - c. IF  $I1==[0/\text{NLIFT}(I)]$  THEN we are at the bounds of the airfoil data
    - i. Set  $I1=[1 /I1-1]$
    - ii.  $I1P1=[2/I1]$
    - iii.  $P1=[0.0/1.0]$
  - d. ELSE regular interpolation indices: set the following
    - i.  $I1P1=I1+1$
    - ii.  $P1= ( \text{AL}(I,I1) - \text{ALPHA} ) / ( \text{AL}(I,I1) - \text{AL}(I,I1P1) );$  interpolation factor
8. IF multiple tables for the current airfoil:  $N\text{Tables}(I)>1$  (*Airfoil*) THEN need to interpolate across tables
- a. Determine the indices for  $\text{MulTabLoc}$ , for an interpolation based on AOA and  $\text{MulTabLoc}$ , and the linear interpolation factor: (note: this could be more efficiently written, minor bug)
    - i. Set  $\text{MulTabLoc } (\text{Airfoil}) = \text{MIN}(\text{MAX}(\text{MulTabLoc}, \text{MulTabMet}(I,1) \text{ (Airfoil)}), \text{MulTabMet}(I,N\text{Tables}(I)) )$  so that it stays within  $\text{MulTabMet}(I,:)$  (e.g.: Reynolds number bounds for instance; (Note this was done at the beginning of this routine inefficiency, minor bug)
    - ii. Call Locatebin to find the index  $I2$  for the value just below the  $\text{MulTabLoc}$  value within the array  $\text{MulTabMet}(I,1:N\text{Tables}(I))$ ; (Note this was done at the beginning of this routine inefficiency, minor bug)
    - iii. IF  $I2==[0/N\text{Tables}(I)]$  THEN we are at the bounds of the airfoil data
      - 1. Set  $I2=[1 /I2-1]$
      - 2.  $I2P1=[2/I2]$
      - 3.  $P2=[0.0/1.0]$  These are the same as those of  $I1 P1$ , inefficient, minor bug
    - iv. ELSE regular interpolation indices: set the following
      - 1.  $I2P1=I2+1$
      - 2.  $P2= ( \text{MulTabLoc} - \text{MulTabMet}(I,I2) ) / ( \text{MulTabMet}(I,I2P1) - \text{MulTabMet}(I,I2) );$  interpolation factor
  - b. Interpolate along AOA first, with fixed Table Parameter:
    - i.  $FSPB = FTB( I,I1,I2P1 ) - (FTB( I,I1,I2P1 ) - FTB( I,I1P1,I2P1 )) * P1;$  this is for Cn f; high table parameter
    - ii.  $FSPA = FTB( I,I1,I2 ) - (FTB( I,I1,I2 ) - FTB( I,I1P1,I2 )) * P1;$  this is for Cn f; low table parameter
    - iii.  $FSPCB = FTBC(I,I1,I2P1) - (FTBC(I,I1,I2P1) - FTBC(I,I1P1,I2P1)) * P1;$  this is for Cc f; high table parameter
    - iv.  $FSPCA = FTBC(I,I1,I2 ) - (FTBC(I,I1,I2 ) - FTBC(I,I1P1,I2 )) * P1;$  this is for Cc f; low table parameter

- v. Interpolate these results for the conclusion of the bilinear interpolation:
1.  $FSP(J,IBLADE) = FSPA + P2 * (FSPB - FSPA)$ ; This is Cn/Cna
  2.  $FSPC(J,IBLADE) = FSPCA + P2 * (FSPCB - FSPCA)$ ; This is Cc/Cna
9. ELSE: no need to interpolate across tables, just AOA (note the minus sign since P1 was defined in the opposite to usual way):
- a.  $FSP(J,IBLADE) = FTB(I,I1,1) - (FTB(I,I1,1) - FTB(I,I1P1,1)) * P1$ ; This is Cn/Cna
  - b.  $FSPC(J,IBLADE) = FTBC(I,I1,1) - (FTBC(I,I1,1) - FTBC(I,I1P1,1)) * P1$ ; This is Cc/Cna
10. IF ABS(  $AFE(J,IBLADE) - AOL1$  ) < 1.E-10: AOA almost at 0-lift THEN: no separation
- a.  $FSP(J,IBLADE) = 1.0$ ;
  - b.  $FSPC(J,IBLADE) = 1.0$
11. ELSE:
- a. Set  $TEMP = 2.*SQRT(ABS(FSP(J,IBLADE)/(AFE(J,IBLADE)-AOL1))) - 1.$ ; this is equivalent to  $\sqrt{f}$  from theory, since  $FSP$  still is at this stage Cn/Cna
  - b. Calculate the actual value with sign of f for Cn:
    - i.  $FSP(J,IBLADE) = TEMP**2 * SIGN(1., TEMP)$ ; f for Cn
    - ii. In case f is >1 or <-1 set it to +1 or -1 respectively
  - c. IF ABS(  $AFE(J,IBLADE)$  ) < 1.E-10: AOA at 0 THEN: from theory the formula would spit out  $C=0$ , so set:
    - i.  $FSPC(J,IBLADE) = 1.0$
  - d. ELSE: revert theory formula for Cc
    - i.  $TEMP = FSPC(J,IBLADE)/((AFE(J,IBLADE)-AOL1)*AFE(J,IBLADE))$ ; this is equivalent to  $\sqrt{f} * \text{recovery factor}$  from theory, since  $FSPC$  still is at this stage Cc/Cna
    - ii. Calculate the actual value with sign of f for Cc:
      1.  $FSPC(J,IBLADE) = TEMP **2 * SIGN(1., TEMP)$ ;  $fc=f$  for Cc (Note the recovery factor is missing)
      2. In case fc is >1 or <-1 set it to +1 or -1 respectively
12. Set  $SRFP = SQRT(ABS(FSP(J,IBLADE))) * SIGN(1., FSP(J,IBLADE)) + 1.$ ; signed square-root of f
13. Set the following ***Bedoes*** variables:
- a.  $FK = 0.25 * SRFP * SRFP$
  - b.  $CVN(J,IBLADE) = CNPOT(J,IBLADE) * (1. - FK)$ ; (increment in vortex lift)

## OUTPUT

It sets the following ***Bedoes (AeroMods [.f90])*** variables:

$CNPOT(J,IBLADE)$ : R(ReKi); Cn under potential flow conditions for the current blade element

$FSP(J,IBLADE)$ : R(ReKi); f for reconstructing Cn for the current blade element

$FSPC(J,IBLADE)$ : R(ReKi); f for reconstructing Cc for the current blade element

$AFE(J,IBLADE)$ : R(ReKi); Effective AOA to be used in the calculation of separation point f (at step 0, it is set=AOA)

*FK: R(ReKi); correction factor for Cn which is also used to calculate CVN*

*CVN(J,IBLADE): R(ReKi); increment in vortex lift*

## C8. BEDDOES(W2, J, IBLADE, ALPHA, CLA, CDA, CMA)

Subroutine that calculates Cl, Cd, Cm via dynamic stall model, regardless of the actual condition of the airfoil. Called by ELEMFR (AeroSubs) from the second time step on. External (Invoked) Modules

- *Airfoil* [see AeroMods.f90]
- *Beddoes* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
W2	R(ReKi)	IN	Relative velocity squared over blade element
J	I	IN	Blade element index
IBlade	I	IN	Blade index
ALPHA	R(ReKi)	IN	AOA
CLA	R(ReKi)	OUT	Cl through LBM model
CDA	R(ReKi)	OUT	Cd through LBM model
CMA	R(ReKi)	OUT	Cm through LBM model

Many additional input via variables in the invoked modules. E.g.: ANE (*Beddoes*).

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
AE	R(ReKi)			AOA at 3/4-chord
AOD1	R(ReKi)			AOA at 0drag interpolated across airfoil tables
AOL1	R(ReKi)			AOA at 0lift interpolated across airfoil tables
CNA1	R(ReKi)			Cn slope coefficient interpolated across airfoil tables
CNS1	R(ReKi)			Cn at upper stall angle
CNSL1	R(ReKi)			Cn at lower stall angle
CDO1	R(ReKi)			Cd0
CA	R(ReKi)			Cos AOA
SA	R(ReKi)			Sin AOA
P	R(ReKi)			Interpolation factor
VREL	R(ReKi)			Air-velocity magnitude relative to blade element

<i>I</i>	I(4)	Index for the airfoil file to use for the <i>j</i> -th blade element
<i>N</i>	I(4)	Index within airfoil
<i>NP1</i>	I(4)	N+1

**STEPS**

1. Set  $I = NFOIL(J)$  (*Airfoil*) , index of airfoil file to use for the  $J$  blade-element under consideration
2. IF multiple tables for the current airfoil:  $NTables(I)>1$  (*Airfoil*) THEN interpolate across tables for  $CNA1, AOL1, CNS1, CNSL1, AOD1, CDO1$  airfoil constants:
  - a. Set  $MulTabLoc$  (*Airfoil*)= MIN( MAX(  $MulTabLoc, MulTabMet(I,1)$  (*Airfoil*)),  $MulTabMet(I,NTables(I))$  ) so that it stays within  $MulTabMet(I,:)$  (e.g.:Reynolds) number bounds for instance; i.e. set the Table Parameter: ( *ElemFrc* (*AeroSubs*) might have set it to the Reynolds Number for instance)
  - c. Call *Locatebin* (*NWTC\_Num* within *NWTC\_Library*) to find the index  $N$  for the value just below the  $MulTabLoc$  value within the array  $MulTabMet(I,1:NTables(I))$  (Note: this and other airfoil data parameters below were read in via *AD\_GetInput/ReadFL* (*AeroSubs*))
  - b. IF  $N==[0/NTables(I)]$  THEN:
    - i. Set  $CNA1 = CNA(I,[1/N])$  (*Bedoes*)
    - ii. Set  $AOL1 = AOL(I, [1/N])$  (*Bedoes*)
    - iii. Set  $CNS1 = CNS(I,[1/N])$  (*Bedoes*)
    - iv. Set  $CNSL1 = CNSL(I,[1/N])$  (*Bedoes*)
    - v. Set  $AOD1 = AOD(I, [1/N])$  (*Bedoes*)
    - vi. Set  $CDO1 = CDO(I, [1/N])$  (*Bedoes*)
  - c. ELSE: regular interpolation to be performed:
    - i. Set  $NP1=N+1$
    - ii. Set  $P=(MulTabLoc-MulTabMet(I,N))/(MulTabMet(I,NP1)-MulTabMet(I,N))$
    - iii.  $CNA1 = CNA(I,N) + P * ( CNA(I,NP1) - CNA(I,N) )$
    - iv.  $AOL1 = AOL(I,N) + P * ( AOL(I,NP1) - AOL(I,N) )$
    - v.  $AOD1 = AOD(I,N) + P * ( AOD(I,NP1) - AOD(I,N) )$
    - vi.  $CNS1 = CNS(I,N) + P * ( CNS(I,NP1) - CNS(I,N) )$
    - vii.  $CNSL1 = CNSL(I,N) + P * ( CNSL(I,NP1) - CNSL(I,N) )$
    - viii.  $CDO1 = CDO(I,N) + P * ( CDO(I,NP1) - CDO(I,N) )$
3. ELSE: no interpolation needed 1 table only, set:
  - a.  $CNA1 = CNA(I,1)$
  - b.  $AOL1 = AOL(I,1)$
  - c.  $AOD1 = AOD(I,1)$
  - d.  $CNS1 = CNS(I,1)$
  - e.  $CNSL1 = CNSL(I,1)$
  - f.  $CDO1 = CDO(I,1)$

4. IF  $CNA1 == 0$  THEN : lift-curve slope is null
  - a. Set  $CLA=0.0$
  - b. Set  $CDA=CD01$
  - c. RETURN
5. Set  $AN$  (*Bedoes*) =  $ALPHA$  ; this is the original AOA
6. Set  $VREL = \sqrt{W2}$ ; this is the air velocity magnitude relative to the blade element
7. Call ATTACH(  $VREL, J, IBlade, CNA1, AOL1, AE$  ) (*AeroSubs*)
8. Call SEPAR(  $NLIFT(I), J, IBlade, I, CNA1, AOL1, CNS1, CNSL1$  ) (*AeroSubs*)
9. Call VORTEX(  $J, IBlade, AE$  ) (*AeroSubs*)
10. Set cos/sin of AOA:
  - a.  $CA = \cos(AN)$
  - b.  $SA = \sin(AN)$
11. Reconstruct output:  $Cl$  and  $Cd$  from  $Cn$  and  $Cc$  of LBM, plus  $Cm$ :
  - a. Set  $CLA = CN * CA + CC * SA$
  - b. Set  $CDA = CN * SA - CC * CA + CD01$ ; Note here we need to add the viscous drag as well
  - c. Set  $CMA = PMC$  (*Airfoil*)

## OUTPUT

$CLA, CDA, CMA$ : R(ReKi);  $Cl, Cd, Cm$  for the current blade element according to LBM model

It also calculates *Bedoes* variables such as:  $CN, CC$ , and sets  $AN$ . It sets *PMC (Airfoil)* via calls to subroutines.

## C9. ATTACH(VREL, J, IBLADE, CNA1, AOL1, AE)

Subroutine that computes the first step of the LBM model. Called by **BEDDOES (AeroSubs)**.

### EXTERNAL (INVOKED) MODULES

- *AeroTime* [see AeroMods.f90]
- *Beddoes* [see AeroMods.f90]
- *Blade* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
VREL	R(ReKi)	IN	Relative air velocity magnitude
J	I	IN	Blade element index
IBlade	I	IN	Blade index
CNA1	R(ReKi)	IN	Cna calculated for the current blade element by <u>Beddoes</u> ( <i>AeroSubs</i> )
AOL1	R(ReKi)	IN	AOL calculated for the current blade element by <u>Beddoes</u> ( <i>AeroSubs</i> )
AE	R(ReKi)	OUT	AOA at 3/4-chord

Many additional input via variables in the invoked modules. E.g.: AN (*Beddoes*), DT (AeroTime) comes from .

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
B2	R(ReKi)			Prandtl-Glauert Beta^2= (1-XM**2)
BS	R(ReKi)			B2 * DS ( <i>Beddoes</i> )
CNI	R(ReKi)			Non-circulatory Cn due to step change in AOA
CNQ	R(ReKi)			Non-circulatory Cn due to step change in pitch-rate
CO	R(ReKi)			This is the theory's T <sub>α</sub> /M
DA	R(ReKi)			Delta ANE (modified AOA) between time steps
PRP	R(ReKi)			Non-dimensional time derivative of ANE
X	R(ReKi)			Non dimensional DT
XKA	R(ReKi)			Factor for the Time Constant T <sub>α</sub>
XM	R(ReKi)			Mach Number (M)

<i>SuperSonic</i>	L	.FALSE.	Initialized/ SAVE	Flag for Supersonic Conditions
-------------------	---	---------	----------------------	--------------------------------

**STEPS**

1. First make a correction to AOA to account for angles larger than pi/2:
  - a. IF ABS(AN (*Bedoes*) ) <= Pi/2 THEN
    - i. Set ANE (J,IBlade) (*Bedoes*) = AN; AN was initially set in BEDDOES (*AeroSubs*)
  - b. ELSEIF AN > Pi/2 THEN
    - i. Set ANE (J,IBlade) = Pi - AN;
  - c. ELSE : AN < -Pi/2
    - i. Set ANE (J,IBlade) = - Pi - AN;
2. Set the Mach Number:  $XM = VREL / AS$  (*Bedoes*)
3. Check if Supersonic:
  - a. IF .NOT. *SuperSonic* .AND.  $XM \geq 1.0$  THEN
    - i. Set  $XM = 0.7$ ; I.e. arbitrarily set Mach number to 0.7
    - ii. Set *SuperSonic* = .TRUE.
    - iii. Warn user
  - b. ELSEIF *SuperSonic* .AND.  $XM < 1.0$  THEN
    - i. Update *SuperSonic* = .FALSE. (since the current conditions are not supersonic)
    - ii. Warn user of conditions back to subsonic
4. Set Prandlt-Glauert beta parameter^2:  $B2 = 1 - XM^{**2}$
5. Set  $ds$ , i.e. the “infinitesimal”(discretized) Delta\_s:  $DS$  (*Bedoes*) =  $2. * DT$  (*AeroTime*) \*  $VREL/C(J)$  (*Blade*)
6. Set  $BS = B2 * DS$
7. Calculate the Time Constant for Attached Flow Response  $T_\alpha = XKA * C(J) / AS$ :
  - a.  $XKA = .75 / ( (1 - XM) + PI(NWTC_Num \text{ within } NWTC_Library) * B2 * XM^{**2} * 0.413 )$ ;  
**(Here it seems that B2 should be SQRT(B2) Bug?)**
  - b.  $X = DT$  (*AeroTime*) / [XKA \* C(J) / AS]; non-dimensional time; note: XKA \* C(J) / AS =  $T_\alpha$  in the theory
  - c.  $CO = XKA * C(J) / AS / XM$ ; equivalent to  $T_\alpha / M$
8. Calculate the delta-AOA and time derivative in ANE terms:
  - a.  $DA = ANE(J,IBLADE) - ANE1(J,IBLADE)$  (*Bedoes*)
  - b.  $ADOT(J,IBLADE)$  (*Bedoes*) =  $DA / DT$ ; this is dimensional
9. Calculate now the saturated(trimmed) value of ADOT going through a normalization and back
  - a.  $PRP = ADOT(J,IBLADE) * C(J) / VREL$ ; (non-dimensional ADOT)
  - b.  $PRP = SAT( PRP, 0.03, 0.1 )$ ; (*AeroSubs*) Apply saturation control if above 0.03, with a slope of .1, i.e. the new value will be 0.1\* the distance of the input and the threshold value.
  - c.  $ADOT(J,IBLADE) = PRP * VREL * C(J)$ ; back to dimensional value
10. Determine the non-circulatory part of  $C_n$ :

- a. Determine the non-circulatory part of Cn, first for the response to step change in AOA:
- The deficiency function  $K_\alpha'$  is:  $DN(J,IBLADE) = OLDDN(J,IBLADE) * EXP(-X) + (ADOT(J,IBLADE) - ADOT1(J,IBLADE)) * EXP(-.5*X)$ ; Note ADOT is  $K_\alpha$  in the theory
  - $CNI = 4. * CO * (ADOT(J,IBLADE) - DN(J,IBLADE))$ ; This is the non-circulatory Cn
- b. Determine the non-circulatory part of Cn, now for the response to step change in pitch rate:
- $QX(J,IBLADE) = (ADOT(J,IBLADE) - ADOT1(J,IBLADE)) * C(J)/VREL/DT$ ; this is the time derivative of the non-dimensional pitching rate
  - The deficiency function  $K_q'$  is:  $DQ(J,IBLADE) = OLDDQ(J,IBLADE)*EXP(-X) + (QX(J,IBLADE) - QX1(J,IBLADE)) * EXP(-.5*X)$ ; (Note this seems incorrect as it uses the same X as for Cn\_AOA, rather it should consider  $T_q < T_\alpha$ )
  - $CNQ = -CO * (QX(J,IBLADE) - DQ(J,IBLADE))$ ; (Note this seems incorrect as it uses the same CO as for Cn\_AOA, rather it should consider  $T_q < T_\alpha$ )
  - Sum the two contributions:
    - $CNIQ = MIN( ABS(CNI+CNQ), 1. ) * SIGN( 1., CNI+CNQ )$ ; it is trimmed at +/-1
11. Determine the contribution to non-circulatory Cm:
- First the contribution to Cm from non-circulatory response to step change in AOA:
    - $CMI = -.25 * CNI$
  - Then the contribution to Cm from non-circulatory response to step change in pitching rate:
    - The deficiency function  $K_q''$  is:  $DQP(J,IBLADE) = DQP1(J,IBLADE) * EXP(-X/XKA) + (QX(J,IBLADE) - QX1(J,IBLADE)) * EXP(-.5*X/XKA)$
    - $CMQ = -.25 * CNQ - (XKA*CO/3.) * (QX(J,IBLADE) - DQP(J,IBLADE))$
12. Determine the circulatory part of Cn at the current time step:
- $XN(J,IBLADE)$  (*Bedoes*) =  $OLDXN(J,IBLADE)$  (*Bedoes*) $*EXP(-.14*BS) + .3*DA*EXP(-.07*BS)$ ; 1<sup>st</sup> deficiency function for AOA at 3/4-chord
  - $YN(J,IBLADE)$  (*Bedoes*) =  $OLDYN(J,IBLADE)$  (*Bedoes*) $*EXP(-.53*BS) + .7*DA*EXP(-.265*BS)$ ; 2<sup>nd</sup> deficiency function for AOA at 3/4-chord
  - $AE$  (*Bedoes*) =  $ANE(J,IBLADE) - YN(J,IBLADE) - XN(J,IBLADE)$ ; effective AOA at 3/4-chord
  - Set  $CNCP = CNA1 * (AE - AOL1)$ ; Cn due to circulatory part for potential (attached) flow
13. Determine the total attached/potential-flow Cn at the current time step as sum of circulatory and non-circulatory parts:
- $CNPOT(J,IBLADE) = CNCP + CNIQ$ ; Total Cn for potential (attached) flow
14. Determine the total attached/potential-flow Cc at the current time step
- $CC = CNPOT(J,IBLADE) * AE$  (this should be put as tan(AE), minor bug)

## OUTPUT

*AE*: R(ReKi): Effective AOA at  $\frac{3}{4}$ -chord (not AOI).

It also sets all the following *Bedoes* parameters: *DS*, *ANE(J,IBLADE)*, *ADOT(J,IBLADE)*, *XN(J,IBLADE)*, *YN(J,IBLADE)*, *CNIQ*, *CNPOT(J,IBLADE)*, *CMI*, *CMQ*, *CC()*: R(ReKi).

## C10. SEPAR(NFT, J, IBLADE, IFOIL, CNA1, AOL1, CNS1, CNSL1)

Subroutine that computes the second step of the LBM model. Called by *BEDDOES* (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

- *Airfoil* [see AeroMods.f90]
- *Beddoes* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
<i>NFT</i>	I(4)	IN	Number of rows (AOAs) in the airfoil data file
<i>J</i>	I	IN	Blade element index
<i>IBLADE</i>	I	IN	Blade index
<i>IFOIL</i>	I(4)	IN	Index of airfoil file
<i>CNA1</i>	R(ReKi)	IN	$Cn_\alpha$ calculated for the current blade element by <u><i>Beddoes</i></u> ( <i>AeroSubs</i> )
<i>AOL1</i>	R(ReKi)	IN	AOL calculated for the current blade element by <u><i>Beddoes</i></u> ( <i>AeroSubs</i> )
<i>CNS1</i>	R(ReKi)	IN	$Cn$ at upper stall angle calculated for the current blade element by <u><i>Beddoes</i></u> ( <i>AeroSubs</i> )
<i>CNSL1</i>	R(ReKi)	IN	$Cn$ at lower stall angle calculated for the current blade element by <u><i>Beddoes</i></u> ( <i>AeroSubs</i> )

Many additional input via variables in the invoked modules. E.g.: *CNCP*, *CNIQ* (*Beddoes*).

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>AFEP</i>	R(ReKi)			$AFF'$ , i.e. a new AOA accounting for unsteady TE separation effects on the $Cm$ calculation
<i>AFF</i>	R(ReKi)			Effective AOA, gives the same unsteady LE pressure under static conditions
<i>CMPA</i>	R(ReKi)			Interpolation of $Cm$ across airfoil AOA data, at high value of Table parameter (e.g. Reynolds number)
<i>CMPB</i>	R(ReKi)			Interpolation of $Cm$ across airfoil AOA data, at low value of Table parameter (e.g. Reynolds number)
<i>FSPA</i>	R(ReKi)			Interpolated value of FTB ( <i>Beddoes</i> ) along AOA for low airfoil Table index

<i>FSPB</i> R(ReKi)	Interpolated value of FTB ( <i>Bedoes</i> ) along AOA for high airfoil Table index
<i>FSPCAR</i> (ReKi)	Interpolated value of FTBC ( <i>Bedoes</i> ) along AOA for low airfoil Table index
<i>FSPCB</i> R(ReKi)	Interpolated value of FTBC ( <i>Bedoes</i> ) along AOA for high airfoil Table index
<i>P1</i> R(ReKi)	Interpolation factor along AOA
<i>P2</i> R(ReKi)	Interpolation factor along airfoil Tables
<i>SRFP</i> R(ReKi)	Signed square-root(f) +1
<i>SRFPC</i> R(ReKi)	Signed square-root(fc)
<i>TEMP</i> R(ReKi)	Temporary factor to calculate f
<i>TFE</i> R(ReKi)	Time constant
<i>I1</i> I(4)	Index of AOAs in interpolation to find Kirchoff's f
<i>I1P1</i> I(4)	I1+1
<i>I2</i> I(4)	Index of Table ID ( <i>MulTabMet (Airfoil)</i> ) in interpolation to find Kirchoff's f
<i>I2P1</i> I(4)	I2+1

### STEPS

1. Set *TFE* = *TF* (*Bedoes*); *TF* time constant was initially set in *BEDDAT (AeroSubs)*
2. Start by calculating the deficiency function DP associated with the delay in Cn:
  - a.  $DPP(J,IBLADE) \text{ (*Bedoes*)} = OLDDPP(J,IBLADE) \text{ (*Bedoes*)}^* \text{ EXP}(-DS/TP) + (CNPOT(J,IBLADE) - CNPOT1(J,IBLADE)) * \text{EXP}(-.5*DS/TP)$ ; Note DS, CNPOT (*Bedoes*) was set in *ATTACH (AeroSubs)*; TP (*Bedoes*) was set in *BEDDAT (AeroSubs)*
3. Now calculate Cn', which accounts for delays on the LE pressure gradient and separation:
  - a.  $CNP(J,IBLADE) = CNPOT(J,IBLADE) - DPP(J,IBLADE)$
4. Now calculate  $\Delta Cn'$ :  $CNPD(J,IBLADE) = CNP(J,IBLADE) - CNP1(J,IBLADE)$  (*Bedoes*)
5. Set SHIFT flag to accelerate vortex lift decay if AOA is decreasing: Note CNPD (less-noisy than AOA) is used to verify whether AOA is increasing or decreasing:
  - a. IF (  $ANE(J,IBLADE) \text{ (*Bedoes*)} * CNPD(J,IBLADE) < 0.$  ) THEN: note that ANE is used in the criteria since we may be in  $(-90^\circ; 0^\circ]$  AOA, thus if CNPD<0 we are increasing |AOA|
    - i.  $SHIFT \text{ (*Bedoes*)} = .TRUE.$  ; AOA is decreasing, accelerate vortex decay
  - b. ELSE
    - i.  $SHIFT = .FALSE.$  ; AOA is not decreasing

- c. (Note that SHIFT should be assigned near TFE (where it is used) below, here it is confusing minor bug)
- 6. Set a new effective AOA  $\alpha$ :
  - a.  $AFF = CNP(J,IBLADE)/CNA1 + AOL1$ ; this will be used for the actual  $f'$  (new f value) calculations
- 7. Take care of possible  $AOA > \pi/2$  as done previously in ATTACH (AeroSubs)
  - a. IF (  $AN \leq PIBY2$  ) THEN
    - i.  $AFE(J,IBLADE) (Bedoes) = AFF$
  - b. ELSEIF (  $AN > PIBY2$  ) THEN
    - i.  $AFE(J,IBLADE) = PI - AFF$
  - c. ELSE
    - i.  $AFE(J,IBLADE) = -PI - AFF$
  - d. ENDIF
  - e. Call MPI2PI (NWTC\_Num NWTC\_Library) (  $AFE(J,IBLADE)$  ), back to  $-pi$  to  $pi$
- 8. Now check if  $AFE$  is within airfoil data table, else abort:
  - a. IF (  $AFE(J,IBLADE) < AL(IFOIL,1) (Airfoil)$  ) .OR. (  $AFE(J,IBLADE) > AL(IFOIL,NLIFT(IFOIL) (Airfoil)$  ) THEN: Warn user and exit program altogether
- 9. Set Interpolation along  $AFE$  to find the Kirchoff's Table values for  $f'$ :
  - a. Set  $AFE$  within boundaries of AOA table:  $AFE(J,IBLADE) = MIN( MAX( AFE(J,IBLADE), AL(IFOIL,1) ), AL(IFOIL,NFT) )$
  - b. Find lower index  $I1$  of AOA arrays (just below  $AFE$ ): Call LocateBin (NWTC\_Num NWTC\_Library)
  - c. IF ( $I1 == 0$ ) THEN
    - i.  $I1 = 1$
  - d. ELSE IF (  $I1 == NFT$  ) THEN
    - i.  $I1 = I1 - 1$  ;
  - e.  $I1P1 = I1 + 1$ ; this is the upper index in the linear interpolation
  - f. Set interpolation factor:  $P1 = ( AL(IFOIL,I1) - AFE(J,IBLADE) ) / ( AL(IFOIL,I1) - AL(IFOIL,I1P1) )$
- 10. IF  $NTables(IFOIL) > 1$  THEN: multiple airfoil tables, Set Interpolation along  $MulTabLoc$  to find the Kirchoff's Table values for  $f'$ :
  - a. (note: this could be more efficiently written, minor bug)
    - i. Set  $MulTabLoc (Airfoil) = MIN( MAX( MulTabLoc, MulTabMet(IFoil,1) (Airfoil), MulTabMet(IFoil,NTables(IFoil)) )$  so that it stays within  $MulTabMet(IFoil,:)$  (e.g.:Reynolds) number bounds for instance;
    - ii. Call Locatebin to find the index  $I2$  for the value just below the  $MulTabLoc$  value within the array  $MulTabMet(IFoil,1:NTables(IFoil))$
    - iii. IF  $I2 == [0:Ntables(I)]$  THEN we are at the bounds of the airfoil data
      - 1. Set  $I2=[1 /I2-1]$
      - 2.  $I2P1=[2/I2]$

3.  $P2=[0.0/1.0]$
- iv. ELSE regular interpolation indices: set the following
1.  $I2P1=I2+1$
  2. Set interpolation factor:  $P2= (MulTabLoc - MulTabMet(IFoil,I2)) / (MulTabMet(IFoil,I2P1) - MulTabMet(IFoil,I2));$
- b. Interpolate along AOA first, with fixed Table Parameter:
- i.  $FSPB = FTB(IFoil,I1,I2P1)$  (*Bedoes*) -  $(FTB(IFoil,I1,I2P1) - FTB(IFoil,I1P1,I2P1)) * P1;$  this is for Cn f'; high table parameter
  - ii.  $FSPA = FTB(IFoil,I1,I2) - (FTB(IFoil,I1,I2) - FTB(IFoil,I1P1,I2)) * P1;$  this is for Cn f'; low table parameter
  - iii.  $FSPCB = FTBC(Foil,I1,I2P1)$  (*Bedoes*) -  $(FTBC(IFoil,I1,I2P1) - FTBC(IFoil,I1P1,I2P1)) * P1;$  this is for Cc f'; high table parameter
  - iv.  $FSPCA = FTBC(IFoil,I1,I2) - (FTBC(IFoil,I1,I2) - FTBC(IFoil,I1P1,I2)) * P1;$  this is for Cc f'; low table parameter
  - v. Interpolate these results for the conclusion of the bilinear interpolation: note that  $FTB$   $FTBC$  were set in **BEDDAT** as Cn/Cna and Cc/Cna
    1.  $FSP(J,IBLADE)$  (*Bedoes*) =  $FSPA + P2 * (FSPB - FSPA);$  This is Cn/Cna
    2.  $FSPC(J,IBLADE)$  (*Bedoes*) =  $FSPCA + P2 * (FSPCB - FSPCA);$  This is Cc/Cna
11. ELSE: no need to interpolate across tables, just AOA (note the minus sign since  $P1$  was defined in the opposite to usual way):
- a.  $FSP(J,IBLADE) = FTB(IFoil,I1,1) - (FTB(IFoil,I1,1) - FTB(IFoil,I1P1,1)) * P1;$  This is Cn/Cna
  - b.  $FSPC(J,IBLADE) = FTBC(IFoil,I1,1) - (FTBC(IFoil,I1,1) - FTBC(IFoil,I1P1,1)) * P1;$  This is Cc/Cna
12. IF  $ABS(AFE(J,IBLADE) - AOL1) < 1.E-10:$  AOA almost at 0-lift THEN: no separation
- a.  $FSP(J,IBLADE) = 1.0;$  f' for Cn
  - b.  $FSPC(J,IBLADE) = 1.0;$  f' for Cc
13. ELSE:
- a. Set  $TEMP = 2.*SQRT(ABS(FSP(J,IBLADE)/(AFE(J,IBLADE)-AOL1))) - 1.;$  this is equivalent to  $\sqrt{f}$  from theory, since  $FSP$  still is at this stage Cn/Cna
  - b. Calculate the actual value with sign of f' for Cn:
    - i.  $FSP(J,IBLADE) = TEMP**2 * SIGN(1., TEMP);$  f' for Cn
    - ii. In case f is >1 or <-1 set it to +1 or -1 respectively
  - c. IF  $ABS(AFE(J,IBLADE)) < 1.E-10:$  AOA at 0 THEN: from theory the formula would spit out  $C=0,$  so set:
    - i.  $FSPC(J,IBLADE) = 1.0;$  fc'=f' for Cc
  - d. ELSE: revert theory formula for Cc
    - i.  $TEMP = FSPC(J,IBLADE)/((AFE(J,IBLADE)-AOL1)*AFE(J,IBLADE));$  this is equivalent to  $\sqrt{f} * \text{recovery factor}$  from theory, since  $FSPC$  still is at this stage Cc/Cna

- ii. Calculate the actual value with sign of  $f'$  for  $C_c$ :
  - iii.  $FSPC(J,IBLADE) = TEMP **2 * SIGN(1., TEMP); fc'=f'$  for  $C_c$  (Note the recovery factor is missing)
  - iv. In case  $fc$  is  $>1$  or  $<-1$  set it to  $+1$  or  $-1$  respectively
14. Now check whether LE separation is occurring:
- a. IF ( $CNP(J,IBLADE) > CNS1$ ) .OR. ( $CNP(J,IBLADE) < CNSL1$ ) THEN:
    - i.  $BEDSEP(J,IBLADE)$  (*Bedoes*)= .TRUE. ;  $BEDSEP$  was initialized at .FALSE. in **BEDDAT** (*AeroSubs*) (poorly written IF in AeroDyn)
15. IF ( $BEDSEP(J,IBLADE)$ ) THEN: Modify  $TAU$  accordingly:
- a.  $TAU(J,IBLADE)$  (*Bedoes*)=  $OLDTAU(J,IBLADE)$  (*Bedoes*) $+ DS/TVL$  ; Note  $DS$  (*Bedoes*) was set in **ATTACH** and  $TVL$ (*Bedoes*) in **BEDDAT** (*AeroSubs*)
16. IF ( $SHIFT$ )  $TFE = 1.5 * TFE$ ; Note  $TFE$  was set at the beginning of the routine equal to  $TF$  (*Bedoes*);  $SHIFT$  indicates that vortex should decay faster
17. Define the following Deficiency functions for  $f''$  and  $fc''$ :
- a.  $DF(J,IBLADE)$  (*Bedoes*)=  $OLDDF(J,IBLADE)$  (*Bedoes*) $* EXP(-DS/TFE) + (FSP(J,IBLADE) - FSP1(J,IBLADE)) * EXP(-.5*DS/TFE)$
  - b.  $DFC(J,IBLADE)$  (*Bedoes*)=  $OLDDFC(J,IBLADE)$  (*Bedoes*)  $* EXP(-DS/TFE) + (FSPC(J,IBLADE) - FSPC1(J,IBLADE)) * EXP(-.5*DS/TFE)$
18. Adjust  $f'$  and  $fc'$  to  $f''$  and  $fc''$ :
- a.  $FP = FSP(J,IBLADE) - DF(J,IBLADE)$ ; this is  $f''$
  - b.  $FPC = FSPC(J,IBLADE) - DFC(J,IBLADE)$ ; this is  $fc''$
19. Calculate the signed square-root of  $f'$  and  $fc'$  to reconstruct  $C_n$  and  $C_c$ :
- a.  $SRFP = SQRT(ABS(FP)) * SIGN(1., FP) + 1.$
  - b.  $SRFPC = SQRT(ABS(FPC)) * SIGN(1., FPC)$
20. Next calculate the new  $C_n$  and  $C_c$  in TE separated flow:
- a. First set a temporary function, which will be used also in **VORTEX** (*AeroSubs*):
  - i.  $FK = 0.25 * SRFP * SRFPC$
  - b.  $CN$  (*Bedoes*)=  $CNCP$  (*Bedoes*) $* FK$  (*Bedoes*) $+ CNIQ$  (*Bedoes*)
  - c.  $CC$  (*Bedoes*) =  $CC$  (*Bedoes*) $* SRFPC$  (No recovery factor here, likely included due to  $fc$  vs  $f$  treatment)
21. Now start calculating the moment coefficient  $C_m$ :
- a. Define a new AOA angle:  $AFEP$ :
  - i.  $DFAFE(J,IBLADE) = DFAFE1(J,IBLADE) * EXP(-DS/(.1*TFE)) + (AFE(J,IBLADE) - AFE1(J,IBLADE)) * EXP(-.5*DS/(.1*TFE))$ ; this is the deficiency function
  - ii.  $AFEP = AFE(J,IBLADE) - DFAFE(J,IBLADE)$
  - b. Interpolate within the airfoil data with this new AOA:
  - i. Set  $AFEP$  within boundaries of AOA table:  $AFEP(J,IBLADE) = MIN( MAX( AFEP(J,IBLADE), AL(IFOIL,1) ), AL(IFOIL,NFT) )$
  - ii. Find lower index  $I1$  of AOA arrays (just below  $AFEP$ ): Call **LocateBin** (*NWTC\_Num NWTC\_Library*)

- iii. IF ( $I1 == [0/NFT]$ ) THEN
  - 1.  $I1 = [1/I1-1]$
  - 2.  $P1=[0.0/1.0]$
- iv. ELSE:
  - 1.  $I1P1 = I1 + 1$ ; this is the upper index in the linear interpolation
  - 2.  $P1 = (AL(IFOIL,I1) - AFEP(J,IBLADE)) / (AL(IFOIL,I1) - AL(IFOIL,I1P1))$ ; this is the interpolation factor
- c. IF ( $NTables(IFOIL) > 1$ ) THEN: multiple airfoil tables:
  - i. Set  $CMPB = CM(IFOIL,I1,I2P1)$  (*Airfoil*) -  $(CM(IFOIL,I1,I2P1) - CM(IFOIL,I1P1,I2P1)) * P1$ ; this is Cm interpolated in AOA at high Table Parameter, note I2P1 was calculated way earlier in the routine
  - ii. Set  $CMPA = CM(IFOIL,I1,I2) - (CM(IFOIL,I1,I2) - CM(IFOIL,I1P1,I2)) * P1$ ; this is Cm interpolated in AOA at low Table Parameter
  - iii. Set  $PMC$  (*Airfoil*) =  $CMPA + P2 * (CMPB - CMPA)$ ; this is Cm interpolated across Tables
- d. ELSE : just one airfoil table
  - i.  $PMC = CM(IFOIL,I1,1) - ((CM(IFOIL,I1,1) - CM(IFOIL,I1P1,1)) * P1)$ ; moment coefficient at  $\frac{1}{4}$  chord

## OUTPUT

$DPP(J,IBLADE)$ ,  $CNP(J,IBLADE)$ ,  $CNPD(J,IBLADE)$ ,  $SHIFT$ ,  $TAU(J,IBLADE)$ ,  $AFE(J,IBLADE)$ ,  $BEDSEP(J,IBLADE)$ ,  $FSP(J,IBLADE)$ ,  $FSPC(J,IBLADE)$ ,  $DF(J,IBLADE)$ ,  $DFC(J,IBLADE)$ ,  $FK$ ,  $CN$ ,  $CC$ : R(ReKi) (*Bedoes*).

It sets  $PMC$ : R(ReKi) (*Airfoil*); the moment coefficient.

It also sets/updates *MulTabLoc* (*Airfoil*): R(ReKi); the current value of the airfoil characteristic to be used for interpolation, e.g.: Reynolds number.

## C11. VORTEX(J,IBLADE,AE)

Subroutine that computes the third step of the LBM model. Called by BEDDOES (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

- *Airfoil* [see AeroMods.f90]
- *Beddoes* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
<i>J</i>	I	IN	Blade element index
<i>IBLADE</i>	I	IN	Blade index
<i>AE</i>	R(ReKi)	IN	Effective AOA at 3/4-chord from <u>ATTACH</u> ( <i>AeroSubs</i> )

Many additional input via variables in the invoked modules. E.g.: *CNCP,DS* (*Beddoes*).

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>CMV</i>	R(ReKi)			LE vortex contribution to 1/4-chord pitching moment
<i>TSH</i>	R(ReKi)			Vortex shedding time constant
<i>TVE</i>	R(ReKi)			Time constant for strength of shed vortex

### STEPS

1. Set *TVE* = *TV* (*Beddoes*); *TV* time constant was initially set in BEDDAT (*AeroSubs*)
2. Set *CVN(J,IBLADE)* (*Beddoes*) = *CNCP* (*Beddoes*); \* ( 1. - *FK(Beddoes)*); Note: *CNCP* was set in ATTACH (*AeroSubs*) and *FK* in SEPAR (*AeroSubs*); This represents the increment toward the accumulated vortex lift
3. IF *TAU(J,IBLADE)* (*Beddoes*) < 1. THEN: this means vortex is not at TE yet; *TAU* was set in SEPAR (*AeroSubs*)
  - a. *VOR* (*Beddoes*) = .TRUE.
  - b. IF (*SHIFT* (*Beddoes*)) THEN *VOR* = .FALSE.; *SHIFT* indicates whether the AOA is decreasing, in which case the vortex should decay faster, no further vorticity accumulation (*SHIFT* set in SEPAR (*AeroSubs*));
4. ELSE : *VOR* = .FALSE.; i.e., vortex at TE, no accumulation of further vorticity needed
5. IF (*VOR*) THEN: vortex contribution to lift is still increasing, add that to normal force contribution from vortex *CNV*:
  - a.  $CNV(J,IBLADE) \quad (Beddoes) = OLDCNV(J,IBLADE) \quad (Beddoes) * EXP(-DS(Beddoes)/TVE) + (CVN(J,IBLADE) - CVN1(J,IBLADE) (Beddoes)) * EXP(-.5*DS/TVE)$
6. ELSE: vortex is just decaying, no further contribution to vortex lift, and decay at twice the rate
  - a.  $CNV(J,IBLADE) = OLDCNV(J,IBLADE) * EXP(-DS/(TVE*.5))$

7. Now calculate new Cn:

$$a. \quad CN(\text{Bedoes}) = CN + CNV(J,IBLADE)$$

8. Now calculate new Cc:

$$a. \quad CC(\text{Bedoes}) = CC + CNV(J,IBLADE) * AE(\text{Bedoes}) * (1. - TAU(J,IBLADE)); \quad AE \text{ is the effective AOA at } \frac{3}{4}\text{-chord set previously in } \underline{\text{ATTACH}} \text{ (AeroSubs)} \text{ (could be written as } \tan(AE) \text{ minor bug)}$$

9. Now take care of the Pitching Moment:

$$a. \quad CMV = -0.2 * (1. - \cos(\pi * TAU(J,IBLADE)) (\text{Bedoes})) * CNV(J,IBLADE); \text{ contribution due to the LE vortex}$$

b. Total Pitching Moment is:  $PMC(\text{Bedoes}) = PMC + CMV + CMI(\text{Bedoes}) + CMQ(\text{Bedoes})$ ; Note CMI and CMQ are non-circulatory responses calculated in ATTACH (AeroSubs), PMC was calculated under unsteady TE-separation from a look-up table of static values in SEPAR(AeroSubs)

10. Take care of the various flags and check whether TAU needs to be reset: (it looks like all of this could be better written minor bug)

a. Set a new time constant:  $TSH = 2. * (1. - FP(\text{Bedoes})) / .19$ ; Note FP is the lagged f', i.e. f'' (SEPAR (AeroSubs)); this constant is based on a vortex shedding Strouhal number of 0.19

b. IF  $TAU(J,IBLADE) .GT. (1. + TSH/TVL) .AND. (.NOT. SHIFT)$  THEN: conditions are ripe for new LE vortex, reset everything

i.  $TAU(J,IBLADE) = 0.$ ; reset TAU, possible new vortex

ii.  $BEDSEP(J,IBLADE)(\text{Bedoes}) = .FALSE.$ ; BEDSEP was set in SEPAR (AeroSubs)

c. IF (  $TAU(J,IBLADE) .GT. 1.$  ) THEN: vortex passed TE, now check if a new cycle may be starting, with an increase in AOA toward stall after reduction in AOA

i. IF (  $ANE(J,IBLADE) .LT. 0.$  ) THEN: AOA < 0

1. IF (  $CNPD(J,IBLADE)(\text{Bedoes}) \leq 0.$  .AND.  $CNPD1(J,IBLADE)(\text{Bedoes}) \geq 0.$  ) .OR.  $ANE1(J,IBLADE)(\text{Bedoes}) > 0$  THEN: The  $\Delta Cn'$  is currently negative and previously positive, or the previous AOA was positive and now negative

a.  $BEDSEP(J,IBLADE) = .FALSE.$

b.  $TAU(J,IBLADE) = 0.$

ii. ELSE: AOA > 0

1. IF (  $CNPD(J,IBLADE) \geq 0.$  .AND.  $CNPD1(J,IBLADE) \leq 0.$  ) .OR.  $ANE1(J,IBLADE) < 0$  THEN: The  $\Delta Cn'$  is currently positive and previously negative, or the previous AOA was negative and now positive

a.  $BEDSEP(J,IBLADE) = .FALSE.$

b.  $TAU(J,IBLADE) = 0.$

## OUTPUT

*CVN(J,IBLADE), CNV(J,IBLADE) , PMC, TAU(J,IBLADE): R(ReKi) (**Bedoes**).*

*BEDSEP(J,IBLADE) : L (**Bedoes**).*

It sets *PMC: R(ReKi) (Airfoil)*; the moment coefficient.

## C12. BEDUPDATE

Subroutine that updates old Beddoes parameters at new time step. Called by *AD CalculateLoads* (*Aerodyn*).

### EXTERNAL (INVOKED) MODULES

- *Beddoes* [see AeroMods.f90]

### INPUT

Input is via variables within invoked module. They are the current time step's values of *Beddoes* variables.

### LOCAL VARIABLES

N/A

### STEPS

1. Update the "#1" and "OLD#" variables as follows:
  - a. *ANE1* = *ANE*
  - b. *ADOT1* = *ADOT*
  - c. *OLDXN* = *XN*
  - d. *OLDYN* = *YN*
  - e. *CNPOT1* = *CNPOT*
  - f. *OLDDPP* = *DPP*
  - g. *FSP1* = *FSP*
  - h. *FSPC1* = *FSPC*
  - i. *OLDTAU* = *TAU*
  - j. *OLDDF* = *DF*
  - k. *OLDDFC* = *DFC*
  - l. *OLDDN* = *DN*
  - m. *OLDCNV* = *CNV*
  - n. *CVN1* = *CVN*
  - o. *CNP1* = *CNP*
  - p. *CNPD1* = *CNPD*
  - q. *OLDSEP* = *BEDSEP*
  - r. *QX1* = *QX*
  - s. *OLDDQ* = *DQ*
  - t. *AFE1* = *AFE*
  - u. *DQP1* = *DQP*
  - v. *DFAFE1* = *DFAFE*

Note: all variables from *Beddoes* module.

### OUTPUT

It sets all the "past" values of *Beddoes* variables to the current time step's values. List of output variables is above (see also *Beddoes*).

## C13. BEDWRT

Subroutine that writes Bedoes parameters to “.opt” file. Called by *ADOUT* (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

- *AD\_IOParams* [see AeroMods.f90]
- *Airfoil* [see AeroMods.f90]
- *Bedoes* [see AeroMods.f90]

### INPUT

Additional Input is provided via module variables (e.g.: CNA(I,K) AOL(I,K) etc.module *Bedoes*)

### LOCAL VARIABLES

A few auxiliary variables needed as counters in DO\_LOOP, or string variables to prepare output strings for the opt file.

Name	Type	Value	PARAMETER/ Initialized	Description
<i>I</i>	I(4)			Counter for airfoil files
<i>K</i>	I(4)			Counter for airfoil Tables
<i>FrMT</i>	C(70)			Format string

### STEPS

1. In a big DO\_LOOP, cycling on a number of airfoil data tables equal to *NTables*(1)
  - a. For each airfoil file it writes the arrays (1:NumFoil, Ntables): CNA, CNS, CNSL, AOL, AOD, CDO (*Bedoes*)
2. It writes on file unit *UnADopt* (*AD\_IOParams*) TVL, TP, TV, TF (*Bedoes*)

### OUTPUT

It just writes to the opt file the Bedoes input variables as read from airfoil data files and certain Bedoes parameters hardwired in the code (see *BEDDAT*).

## C14. DISKVEL ()

Subroutine that calculates the mean velocities relative to the rotor disk; it calls a routine to get wind velocity at a specified location. Called by *AD\_CalculateLoads* (*Aerodyn*).

### EXTERNAL (INVOKED) MODULES

- *AeroTime* [see AeroMods.f90]
  - Time: R(ReKi)
- *InflowWind* [see AeroMods.f90]
- *Rotor* [see AeroMods.f90]
- *Switch*[see AeroMods.f90]
- *Wind* [see AeroMods.f90]

### INPUT

Input is provided via invoked module variables, e.g.: *CYaw* (*Rotor*).

### LOCAL VARIABLES

A few auxiliary variables needed as counters in DO\_LOOP, or string variables to prepare output strings for units in the headings of the opt file.

Name	Type	Value	PARAMETER/ Initialized	Description
<i>Vinplane</i>	R(ReKi)			Wind velocity component in the plane of rotor
<i>VXY</i>	R(ReKi)			Component normal to rotor plane accounting for yaw and wind direction (not tilt)
<i>AvgInfVel(3)</i> R(ReKi)				Components along inertial FF0 of wind velocity at time and of interest
<i>Position(3)</i>	R(ReKi)			(0.0 , 0.0, <i>HH (Rotor)</i> ) in FF0
<i>ErrStat</i>	I			Error Status Flag

### STEPS

1. Set *Position* = (0.0 , 0.0, *HH (Rotor)*)
2. Set *AvgInfVel(:)=WindInf ADhack\_diskVel(* REAL(*Time*, ReKi), *Position*, *ErrStat* *) (InflowWind)*
3. Calculate horizontal component of mean wind normal to rotor plane assumed vertical (no tilt yet)  
accounting for wind direction (from previous call) and yaw:
  - a.  $VXY = AvgInfVel(1) * CYaw (\textbf{Rotor}) - AvgInfVel(2) * SYaw (\textbf{Rotor})$
4. Calculate components of mean wind speed along x, y, z of rotor plane (note tilt negative if upwind rotor tilted upward), reference frame is the shaft reference frame FFs
  - a.  $VROTORX (\textbf{Wind}) = VXY * CTILT (\textbf{Rotor}) + AvgInfVel(3) * STILT (\textbf{Rotor})$ ; along hub x-axis

- b.  $VROTOR Y (Wind) = AvgInfVel(1) * SYaw + AvgInfVel(2) * CYaw + YAWVEL (Rotor)$ ; along shaft y-axis; YAWVEL and tilt STILT, CTILT calculated in **AD\_CalculateLoads** (**Aerodyn**)
  - c.  $VROTOR Z (Wind) = -1.* VXY * STILT + AvgInfVel(3) * CTILT$ ; along shaft z axis (upward)
5. IF not dynamic inflow (generalized wake) THEN:
- a.  $Vinplane = \text{SQRT}( VROTOR Y * VROTOR Y + VROTOR Z * VROTOR Z )$ ; wind component in the plane of rotor
  - b. IF ( $Vinplane \geq 1.0E-3$ ) THEN
    - i.  $SKEW (Switch) = \text{TRUE}$ .
    - ii.  $SDEL (Wind) = VROTOR Y / Vinplane$ ; sin of angle between Vinplane and horizontal y-hub-axis
    - iii.  $CDEL (Wind) = -VROTOR Z / Vinplane$ ; cos of angle between Vinplane and horizontal y-hub-axis
    - iv.  $ANGFLW (Wind) = \text{ATAN2}(\text{ABS}( VROTOR X - AVGINFL(Rotor) ), Vinplane )$ ; angle between rotor plane and wind velocity; AVGINFL is the previous step average induced velocity
  - c. ELSE  $SKEW = \text{FALSE}$ .

## OUTPUT

It calculates components of wind velocity in the rotor frame of reference accounting for YawVel as well, and sets other variables such as *SKEW*, *SDEL*, *CDEL*, *ANGFLW*, *VROTORX*, *VROTORY*, *VROTORZ* (*Wind*).

## C15. AD\_WINDVELOCITYWITHDISTURBANCE(INPUTPOSITION, ERRSTAT)

Function that computes the (dimensional) wind velocity components at the location *InputPosition* in the inertial frame of reference, including any tower shadow deficit. Called by *AD\_CalculateLoads* (*Aerodyn*).

### EXTERNAL (INVOKED) MODULES

Only a few variables extracted from the following modules:

- *AeroTime* [see AeroMods.f90]
  - *Time*: R(DbKi)
- *Rotor* [see AeroMods.f90]
  - *HH*: R(ReKi)
- *TwrProps* [see AeroMods.f90]
  - *PJM\_Version*: L
  - *TShadC1*: R(ReKi)
  - *TShadC2*: R(ReKi)

### INPUT

Name	Type	INTENT	Description
<i>InputPosition</i> (3)R(ReKi)		IN	X,Y,Z where to calculate wind velocity; i.e. blade element position
<i>ErrStat</i>	I	OUT	Returns 0 if no errors were encountered; non-zero otherwise

Additional input via variables in the invoked modules and modules invoked in the parent module *AeroSubs*.

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>InflowVel</i>		<b>InflIntrpOut</b> <i>(SharedInflowDefns)</i>		Wind Velocity at point of interest
<i>angle</i>	R(ReKi)			Absolute difference between <i>theta</i> and <i>phi</i>
<i>phi</i>	R(ReKi)			Angle between FF0 x-axis and instantaneous horizontal wind direction
<i>dist</i>	R(ReKi)			Distance from blade element to wake centerline
<i>RADIUS</i>	R(ReKi)			Horizontal distance from tower to blade element: in actuality, it's the distance from the undeflected tower centerline, not the

		actual tower
<i>ROOTR</i>	R(ReKi)	SQRT( <i>RADIUS</i> )
<i>SHADO</i> <i>W</i>	R(ReKi)	
<i>TEMP</i>	R(ReKi)	
<i>theta</i>	R(ReKi)	Angle between FF0 x-axis and line from tower to blade element (projected onto (x,y))
<i>width</i>	R(ReKi)	Half width of the wake after accounting for wake expansion proportional to square root of <i>RADIUS</i>
<i>Sttus</i>	I	Error Status Flag

**STEPS**

1. Get undisturbed velocity:
  - a. *InflowVel* = WindInf\_GetVelocity(REAL(Time, ReKi), *InputPosition*, *Sttus*) (*InflowWind* invoked in parent module *AeroSubs*)
2. Set output *AD\_WindVelocityWithDisturbance*(:) = *InflowVel*%Velocity(:)
3. IF *PJM\_Version* THEN calculate tower effects via potential and wake deficit models
  - a. Call GetTwrInfluence(*AD\_WindVelocityWithDisturbance*(1), *AD\_WindVelocityWithDisturbance*(2), *InputPosition*(:)) (module *AeroSubs*)
4. ELSE (old tower treatment): Apply tower shadow if the blade element is in the wake
  - a. IF *TShadC2* (*TwrProps*)> 0.0 (note AD\_GetInput (*Aerosubs*) had initialized the variable) THEN: Perform calculations only if the wake strength is positive
    - i. IF horizontal velocity<>0: IF *AD\_WindVelocityWithDisturbance*(1:2)<>0 THEN
      1. Calculate angle *phi* between X axis and wind direction as ATAN2(*AD\_WindVelocityWithDisturbance*(2), *AD\_WindVelocityWithDisturbance*(1))
      2. Calculate angle *theta* between X axis and projection of line connecting tower centerline to element of interest as: *theta*= ATAN2(*InputPosition*(2), *InputPosition*(1))
      3. Calculate *angle* as difference between *phi* and *theta*
      4. Put *angle* between -Pi and Pi, then take its ABS value
      5. IF *angle* <= Pi/2 THEN: (take care of downwind-of-tower portion only)
        - a. Calculate *radius* = SQRT(*InputPosition*(1)\*\*2 + *InputPosition*(2)\*\*2), distance from tower axis
        - b. Set *RootR*=SQRT(*radius*)
        - c. Set Halfwidth *width*= *RootR* \* *TShadC1* (*TwrProps*) (note AD\_GetInput (*Aerosubs*) had initialized the variable)

- d. IF  $width > 0$  THEN
  - i. Calculate distance of element from freestream centerline  
 $dist = radius * \sin(angle)$
  - ii. IF above tower: InputPosition(3) > HH THEN  $dist = \sqrt{dist^2 + (InputPosition(3) - HH)^2}$ ; this again tries to maintain a continuous function above tower (see also *GetTwrInfluence AeroSubs*) with a 3D wake, semicircular
  - iii. IF element in wake:  $width > dist$  THEN calculate deficit factor (Powles model):
  - iv.  $shadow = TshadC2 / \sqrt{R} * \cos^2(\pi/2 * dist/width)$
  - v.  $AD\_WindVelocityWithDisturbance(1:2) = AD\_WindVelocityWithDisturbance(1:2) * (1. - shadow)$

## OUTPUT

*AD\_WindVelocityWithDisturbance(3)*: R(ReKi). Wind Velocity components in inertial reference frame at location inputted in the call, affected by tower influence.

## C16. GETTWRINFLUENCE(VX, VY, INPUTPOSITION )

Subroutine that computes tower shadow or dam influence on the blade. Note that this routine assumes there are NO tower deflections.

This function should return the influence parameters, which will be based on some mean wind direction (or possibly the direction at the tower) for a given height, instead of using the local velocity/direction so that all points on a horizontal slice of air can use the same deficit at each given time. Will need the tower position, too.

Called by *AD\_WindVelocityWithDisturbance* (*AeroSubs*).

### THEORY

It uses the Riso tower dam model of Bak, Madsen and Johansen. This model is based on potential flow and is applicable in front of and behind the tower, although in the wake the method of Powles, PJM, NREL, is used.

### EXTERNAL (INVOKED) MODULES

Only a few variables extracted from the following modules:

- *Rotor* [see *AeroMods.f90*]
  - *HH*: R(ReKi)
- *TwrProps* [see *AeroMods.f90*]

### INPUT

Name	Type	INTENT	Description
VX	R(ReKi)	INOUT	On input, U-velocity without tower effect; on output, U-velocity including tower effect
VY	R(ReKi)	INOUT	On input, V-velocity without tower effect; on output, V-velocity including tower effect
<i>InputPosition</i> (3)	R(ReKi)	IN	Global coordinates of point of interest where tower effects need to be calculated

Additional input via variables in the invoked modules and modules invoked in the parent module *AeroSubs*.

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
AFLAG	L	.FALSE.	initialized	Set to .TRUE. on possible tower strike
ANGLE	R(ReKi)			Angle determining whether blade is upwind or downwind of tower

<i>CenterDist</i>	R(ReKi)	Distance from blade element to wake centerline
<i>phi</i>	R(ReKi)	Angle between x-axis and horizontal wind direction based upon instantaneous velocities VY and VX (at the blade element)
<i>CosPhi</i>	R(ReKi)	COS( <i>phi</i> )
<i>SinPhi</i>	R(ReKi)	SIN( <i>phi</i> )
<i>Distance</i>	R(ReKi)	Normalized horizontal distance from tower to blade element
<i>SHADOW</i>	R(ReKi)	Value of the tower shadow deficit at the blade element
<i>THETA</i>	R(ReKi)	Angle between x-axis and line from tower to blade element
<i>TwrCD_Station</i>	R(ReKi)	Drag coefficient of the tower
<i>TwrRad</i>	R(ReKi)	Radius of the tower at the height of interest
<i>WIDTH</i>	R(ReKi)	Half width of the wake after accounting for wake expansion proportional to square root of RADIUS
<i>V_total</i>	R(ReKi)	Total freestream wind speed
<i>VX_wind</i>	R(ReKi)	Tower influenced wind speeds in wind coordinates
<i>VY_wind</i>	R(ReKi)	Tower influenced wind speeds in wind coordinates
<i>WindXInf</i>	R(ReKi)	Influence of the tower on X wind velocity in wind reference frame
<i>WindYInf</i>	R(ReKi)	Influence of the tower on Y wind velocity in wind reference frame
<i>Xtemp</i>	R(ReKi)	Temporary variable used in tower dam calculations
<i>Xtemp2</i>	R(ReKi)	Temporary variable used in tower dam calculations

$Xwind$	R(ReKi)	X Location of element in a wind-based coordinate system
$Yg$	R(ReKi)	Variable used to smooth dam effect above the tower
$Ytemp2$	R(ReKi)	Temporary variable used in tower dam calculations
$Ywind$	R(ReKi)	Y Location of element in a wind-based coordinate system
$ZGrnd$	R(ReKi)	Distance between position and undeflected hub

## STEPS

1. Make sure that either *TwrPotent* or *TwrShadow* (*TwrProps*) are .TRUE. ELSE RETURN
2. Initialize  $ZGrnd = InputPosition(3) - HH$  ; Z-distance between position and hub
3. Initialize  $V\_Total = \text{SQRT}(VX^2 + VY^2)$  ; horizontal wind speed
4. If  $V\_Total \leq 0.0$  THEN RETURN (no need for tower effects)
5. Get the tower Cd (*TwrCD\_Station*) and Radius (*TwrRad*) properties for the current element location corresponding to the Z-coordinate of point of interest: Call *GetTwrSectProp* (*InputPosition(:)*, *V\_total*, *TwrRad*, *TwrCD\_Station*) (*AeroSubs*)
6. Calculate normalized distance of point of interest in (X,Y) plane:  $Distance = \text{SQRT} (InputPosition(1)^2 + InputPosition(2)^2) / TwrRad$
7. Check whether  $Distance < 1.0$ , which may indicate tower strike if  $ZGrnd < 0.$ , and warn user and exit
8. Store angle between wind direction and X-axis:  $\phi = \text{ATAN2}(VY, VX)$  and calculate  $\cos\phi$   $\sin\phi$
9. IF *TwrPotent* (*TwrProps*) THEN Calculate the influence due to potential flow around tower based on velocity at element (Bak et al. (2001) Model):
  - a. IF element of interest above tower top:  $ZGrnd > 0$ . THEN smooth transition to free-stream  $Yg = \text{SQRT}(InputPosition(2)^2 + ZGrnd^2)$ :  $Yg$  becomes the radial distance in case we are above the tower to account for a 3D semicircular effect  
ELSE  $Yg = InputPosition(2)$
  - b. Put element location in wind reference frame (note this is not done for the Tower Shadow part though, where FF0 is used):
    - i.  $Xwind = InputPosition(1) * \cos\phi + Yg * \sin\phi$
    - ii.  $Ywind = Yg * \cos\phi - InputPosition(1) * \sin\phi$
  - c. Normalize coordinates by the local element tower radius:  $Xwind = Xwind / TwrRad$  and  $Ywind = Ywind / TwrRad$
  - d. Add *Tower\_wake\_Constant* (*TwrProps*):  $Xtemp = Xwind + Tower\_Wake\_Constant$  (0.1 for Bak model generally, but it is an input in tower file)
  - e. Assign  $Xtemp2 = Xtemp^2$  and  $Ytemp2 = Ywind^2$

- f. Calculate tower influence (Bak model):
- $WindXInf = 1.0 - (Xtemp2 - Ytemp2)/(Xtemp2 + Ytemp2)^{**2} + TwrCD_Station/TwoPi * Xtemp/(Xtemp2 + Ytemp2)$
  - $WindYInf = -2.0 * (Xtemp * Ywind)/(Xtemp2 + Ytemp2)^{**2} + TwrCD_Station/TwoPi * Ywind/(Xtemp2 + Ytemp2)$
10. ELSE (no potential model to be used) set  $WindXInf=1.0$  and  $WindYInf=0.0$
11. IF  $TwrShadow$  (*TwrProps*) THEN Calculate the influence of tower shadow if user specifies and we are downwind of the tower:
- Calculate angle theta, between X-axis and horizontal projection of line from tower center to element of interest to figure out whether we are in the downwind wake of tower:  $theta=ATAN2( InputPosition(2), InputPosition(1) )$
  - Calculate difference in ABS of theta and phi:  $angle= ABS(theta - phi)$  and put as  $-Pi$  and  $Pi$
  - IF  $ABS(angle) < Pi/2$  THEN downwind region
    - Calculate normalized (with tower element radius) wake halfwidth at point of interest from Powles (1983) model:  $width = \sqrt{Distance}$
    - Calculate distance from free-stream centerline  $CenterDist = Distance * \sin(\angle)$
    - IF above tower top:  $ZGrnd>0.0$  THEN  $CenterDist = \sqrt{CenterDist^{**2} + ZGrnd^{**2}}$ ; i.e., considering 3D wake with semicircle cross section from tower-top upward and downwind.
    - IF within wake:  $CenterDist < width$  THEN calculate Powles wake factor
      - $shadow = \cos^2(\pi/2 * CenterDist / width) * TwrCD_Station / width$
      - $WindXInf = 1.0 - shadow$
    - ELSE no velocity correction needed
12. Apply tower effects to input wind speeds, remembering that tower effects are effectively calculated as normalized by the wind speed and in a wind based reference frame, thus that there is also a need to go back to main reference frame:
- $VX\_wind = WindXInf * V\_total$  and  $VY\_wind = WindYInf * V\_total$
  - $VX = VX\_wind * \cos\phi - VY\_wind * \sin\phi$
  - $VY = VY\_wind * \cos\phi + VX\_wind * \sin\phi$

## OUTPUT

$VX, VY$ : R(ReKi). Horizontal Wind Velocity components in inertial reference frame at location inputted in the call corrected by tower effects.

## C17. GETTWRSECTPROP(INPUTPOSITION, VELHOR, TWRELRAD, TWRELCD)

Subroutine that returns the tower radius and Cd, for the vertical location of the element currently being evaluated for tower influence. Called by *GetTwrInfluence* (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

- *Rotor* [see AeroMods.f90]
  - *HH*: R(ReKi), only
- *TwrProps* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
<i>InputPosition</i> (3)R(ReKi)		IN	Location where tower properties are desired
<i>VelHor</i>	R(ReKi)	IN	The horizontal wind speed, used to get Reynolds number, if necessary
<i>TwreRad</i>	R(ReKi)	OUT	Radius of the tower element
<i>TwreCD</i>	R(ReKi)	OUT	Drag coefficient of the tower element

Additional input via variables in the invoked module.

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>P1</i>	R(ReKi)			Interpolation weighting factor
<i>P2</i>	R(ReKi)			Interpolation weighting factor
<i>TwreCD1</i>	R(ReKi)			Dummy variable for 2-D interpolation
<i>TwreCD2</i>	R(ReKi)			Dummy variable for 2-D interpolation
<i>TwreHt</i>	R(ReKi)			Non-dimensional height of the tower element
<i>TwreRe</i>	R(ReKi)			Reynold's # of the tower element
<i>N1</i>	I			Index position in table for interpolation
<i>N2</i>	I			Index position in table for interpolation
<i>N1P1</i>	I			Index position+1 in table for interpolation
<i>N2P1</i>	I			Index position+1 in table for interpolation

### STEPS

1. Set *TwreHt*= *InputPosition*(3) / *HH*; fraction of tower height (perhaps instead of HH a tower position and length should be used)

2. Interpolate to find  $TwrElRad = 0.5 * \text{InterpBin}(TwrElHt, TwrHtFr, TwrWid, N2, NTwrHt)$  ( $NWTC\_Num$  in  $NWTC\_Library$ ) variables from  $TwrProps$ , except the last two that are local variables to this routine
3. Analogously interpolate to find the element  $TwrElCD$ :
  - a. IF only 1 Reynolds row:  $NTwrRe == 1$  THEN
    - i. IF only 1 CD column:  $NTwrCD == 1$  THEN  $TwrElCD = TwrCD(1,1)$
    - ii. ELSEIF  $NTwrHt == 1$  THEN  $TwrElCD = TwrCD(1, NTwrCDCol(1))$ ; this case represents the existence of only 1 tower element in the tower file, therefore even if we have more CD columns, the column number to use is the one indicated by  $NTwrCDCol(1)$  only element)
    - iii. ELSE interpolate to find  $TwrElCD = \text{InterpStp}(TwrElHt, TwrHtFr, TwrCD(1,:), N2, NTwrHt)$  ( $NWTC\_Num$  in  $NWTC\_Library$ )
  - b. ELSE (multiple rows of Reynolds numbers):
    - i. First calculate Reynolds Number (in millions) for the element:  $TwrElRe = \text{GetReynolds}(VelHor, 2.0 * TwrElRad)$  (*Aerosubs*)
    - ii. IF only 1 CD column:  $NTwrCD == 1$  THEN interpolate  $TwrElCD = \text{InterpBin}(TwrElRe, TwrRe, TwrCD(:,1), N1, NTwrRe)$
    - iii. ELSEIF  $NTwrHt == 1$  THEN  $TwrElCD = \text{InterpBin}(TwrElRe, TwrRe, TwrCD(:, NTwrCDCol(1)), N1, NTwrRe)$ ; this case represents the existence of only 1 tower element in the tower file, therefore even if we have more CD columns, the column number to use is the one indicated by  $NTwrCDCol(1)$  only element)
    - iv. ELSE (multiple CD columns and multiple Tower elements in the data file) carry out nearest-neighbor interpolation interpolation:
      1. Call  $\text{LocateBin}(TwrElRe, TwrRe, N1, NTwrRe)$  ( $NWTC\_Num$  in  $NWTC\_Library$ ) (extra variable from  $TwrProps$ ); it returns the lower bound index  $N1$  for value  $TwrElRe$  within array  $TwrRe$
      2. Use usual NREL way to interpolate based on tower element radius, by making use of max and min combined, and return  $TwrElCD$  for the element

## OUTPUT

It returns  $TwrElRad$  and  $TwrElCD$ : R(ReKi) for the selected element of the tower by interpolating tower file data.

## C18. GETREYNOLDS (WINDSPED, CHORDLEN)

Function that returns the Reynolds Number/ $10^6$  for an element. Called by: *ElemFrc* and *GetTwrSectProp* (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

Only a few variables extracted from the following modules:

- *Wind* [see AeroMods.f90]
  - *KinVisc*: R(ReKi)

### INPUT

Name	Type	INTENT	Description
<i>WindSpd</i>	R(ReKi)	IN	Wind Speed
<i>ChordLen</i>	R(ReKi)	IN	Chord/diameter length scale

Additional input via variables in the invoked modules;

### LOCAL VARIABLES

N/A

### STEPS

1. From Definition:  $GetReynolds = 1.0E-6 * WindSpd * ChordLen / KinVisc$

### OUTPUT

*GetReynolds*: R(ReKi). Reynolds number (in millions) for the element (either tower or blade). It is used for interpolation of tower/blade file data, where the Reynolds are given in Re/ $10^6$ .

## C19. AERO<sub>DYN</sub>\_TERMINATE

Subroutine that cleans up at the end of the aerodyn program.

### EXTERNAL (INVOKED) MODULES

- *AD\_IOParams* [see AeroMods.f90]
- *Airfoil* [see AeroMods.f90]
- *Bedoes* [see AeroMods.f90]
- *Blade* [see AeroMods.f90]
- *DynInflow* [see AeroMods.f90]
- *Element* [see AeroMods.f90]
- *EIOutParams* [see AeroMods.f90]
- *ElemInflow* [see AeroMods.f90]
- *TwrProps* [see AeroMods.f90]

### INPUT

Additional Input is provided via invoked module variables.

### LOCAL VARIABLES

N/A

### STEPS

1. IF allocated THEN it will deallocate variables such as
  - a. *FOILNM, AL, CD, CL, CM, MulTabMet, NFOIL, NLIFT, NTables*, (Note *FOILNM* deallocated twice possible minor bug) (*Airfoil*)
  - b. *ADOT, ADOT1, AFE, AFE1, ANE, ANE1, AOD, AOL, CDO, CAN, CNP, CNP1, CNPD, CNPD1, CNPOT, CNPOT1, CNS, CNSL, CNV, CVN, CVN1, DF, DFAFE, DFAFE1, DFC, DN, DPP, DQ, DQP, DQP1, FSP, FSP1, FSPC, FSPC1, FTB, FTBC, OLDCNV, OLDDF, OLDDFC, OLDFN, OLDDPP, OLDDQ, OLDTAU, OLDXN, OLDYN, QX, QX1, TAU, XN, YN, BEDSEP, OLDSEP* (*Bedoes*)
  - c. *C, DR* (*Blade*)
  - d. *RMC\_SAVE, RMS\_SAVE* (*DynInflow*)
  - e. *A, AP, HLCNST, RELM, TLCNST, TWIST* (*Element*)
  - f. *AAA, AAP, ALF, CDD, CLL, CMM, CNN, CTT, DFNSAV, DFTSAV, DynPres, PITSAV, PMM, ReyNum, SaveXY, SaveVY, SaveVZ, WndElPrList, WndElPrNum, ElPrList, ElPrNum* (*EIOutParams*)
  - g. *W2, ALPHA* (*ElemInflow*)
  - h. *TwrHtFr, TwrWid, TwrCD, TwrRe, NTwrCDCol* (*TwrProps*)
2. Close the following file units:
  - a. *UnADin* ("ipt", input file) (*AD\_IOParams*)
  - b. *UnADopt* ("opt", echo of options selected file) (*AD\_IOParams*)
  - c. *UnAirfl* (airfoil data file) (*AD\_IOParams*)
  - d. *UnWind* (wind data file, FF or HH) (*AD\_IOParams*)
  - e. *UnEc* (Echo file) (*NWTC\_Library*)

## OUTPUT

It just deallocates variables from various modules and closes files.

## C20. ABPRECOR(F, OLDF, DFDT, DT, N, NO)

Subroutine that integrates function  $F$  by Adams-Bashforth Predictor and Adams-Moulton Corrector using 4 previous values of DF/Dt. Called by *InfDist* (*AeroSubs*).

### THEORY

The Adams-Bashforth Predictor Corrector Method is based on a 4<sup>th</sup> order approximation via Lagrange polynomial of the integrand function (DFDT), and as such it needs 4 previous points of DFDT. Integrating Lagrange polynomial gives the Predictor formula.

The Corrector Step uses the Predictor result to obtain yet a new point for DFDT, and then re-uses Lagrange approximation polynomial across the last 4 points of DFDT. Integrating that gives the Corrector formula.

### EXTERNAL (INVOKED) MODULES

N/A

### INPUT

Name	Type	INTENT	Description
$N$	I(4)	IN	Final index for $DFDT$ and $F$
$N0$	I(4)	IN	Starting index for $DFDT$ and $F$
$DT$	R(ReKi)	IN	Time Step
$DFDT(N0:N, 4)$	R(ReKi)	IN	Time Derivative Values of $F$
$F(N0:N)$	R(ReKi)	OUT	The function values to be calculated from the knowledge of $DFDT$
$OLDF(N0:N)$	R(ReKi)	IN	Previous time step values of $F$

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
$DFDT0$	R(ReKi)			Intermediate time derivative for corrector step
$I$	I			counter

### STEPS

1. In a DO\_LOOP cycle from  $I=N0$  to  $N$ :
  - a. Predictor Step:  $F(I) = OLDF(I) + ( 55. * DFDT(I,1) - 59. * DFDT(I,2) + 37. * DFDT(I,3) - 9. * DFDT(I,4) ) * DT / 24.$
  - b. New time derivative  $DFDT0 = ( F(I) - OLDF(I) ) / DT$

- c. Corrector Step:  $F(I) = OLDF(I) + ( 9. * DFDT0 + 19. * DFDT(I,1) - 5. * DFDT(I,2) + DFDT(I,3) ) * DT / 24.$

## OUTPUT

It returns  $F(N0:N)$ : R(ReKi). Values of F integrated.

## C21. ELEMFRC (PSI, RLOCAL, J, IBLADE, VNROTOR2, VT, VNW, VNB, DFN, DFT, PMA, INITIAL)

Subroutine that calculates the aerodynamic forces on one blade elements. It is called by [AD\\_CalculateLoads \(Aerodyn\)](#).

### THEORY

BEM and Generalized Dynamic Wake

### EXTERNAL (INVOKED) MODULES

- *ElOutParams* [see AeroMods.f90]
- *Airfoil* [see AeroMods.f90]
- *Blade* [see AeroMods.f90]
- *Element* [see AeroMods.f90]
- *ELEMInflow* [see AeroMods.f90]
- *InducedVel* [see AeroMods.f90]
- *Rotor* [see AeroMods.f90]
- *Switch* [see AeroMods.f90]
- *Wind* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
VNROTOR2R(ReKi)	IN		Square of the wind velocity normal to the rotor plane
VNB	R(ReKi)	IN	Component of the translational velocity normal to plane of rotor
VNW	R(ReKi)	IN	Component of wind velocity normal to rotational plane (rotor)
VT	R(ReKi)	INOUT	Tangential velocity accounting for wind and rotation/deflections of blade
PSI	R(ReKi)	IN	Azimuth angle of blade under analysis [rad]
RLOCAL	R(ReKi)	IN	Local radial distance of element in the plane of rotation
J	I	IN	Blade element counter
IBlade	I	IN	Blade counter
Initial	L	IN	First pass or not at load calculation for that blade element
DFN	R(ReKi)	OUT	Blade element force normal to plane of rotation
DFT	R(ReKi)	OUT	Blade element force tangent to plane of rotation
PMA	R(ReKi)	OUT	Blade element torque

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>ErrStat</i>	I			Error Status Flag
<i>CDA</i>	R(ReKi)			Blade Element Cd (including possible induction effects)
<i>CLA</i>	R(ReKi)			Blade Element Cl (including possible induction effects)
<i>CMA</i>	R(ReKi)			Blade Element Torque Moment (including possible induction effects)
<i>CPHI</i>	R(ReKi)			$\cos(\phi)$
<i>PHI</i>	R(ReKi)			Angle of attack if no pitch is present
<i>SPHI</i>	R(ReKi)			$\sin(\phi)$
<i>QA</i>	R(ReKi)			Aero-force base factor: $0.5 * \rho * W^2 * C * DR$
<i>ReNum</i>	R(ReKi)			Local Reynolds Number in millions
<i>Vinduced</i>	R(ReKi)			Induced velocity
<i>VN</i>	R(ReKi)			Effective relative velocity normal to the plane of rotation

## STEPS

1. IF  $RLOCAL < 1$  cm:
  - a. set  $A(J, IBlade) = AP(J, IBlade) = 0.0$  (*Element*) ; don't bother calculating induction factors
2. ELSEIF *DYNINFL* (*Switch*) .AND. ( $R(Blade)^* REVS(Rotor) < 2.0$  m/s) THEN: this tries to take care of problems with dynamic inflow at low tip speed
  - a. set  $A(J, IBlade) = AP(J, IBlade) = 0.0$  (*Element*),
  - b. set *DYNINIT*=.TRUE. (*Switch*)
3. ELSE:
  - a. IF (*WAKE (Switch)* .AND. .NOT. *Initial*) THEN: (i.e. if axial induction factor is to be calculated and we are not at the beginning) Get induction factor =  $A$  using static airfoil coefficients:
    - i. IF *DYNINFL* THEN:
      1. Use dynamic inflow model to find  $A$ : Call *VINDINF*( *J, IBlade, RLOCAL, VNW, VNB, VT, PSI* ) (*AeroSubs*)
    - ii. ELSE: use BEM to find  $A$ :
      1. Call *VIND*( *J, IBlade, RLOCAL, VNROTOR2, VNB, VT* ) (*AeroSubs*)
      2. IF *SKEW (Switch)* THEN Call *VNMOD*( *J, IBlade, RLOCAL, PSI* ) (*AeroSubs*), i.e. apply skewed-wake correction

- b. ELSE (induced velocity will be ignored if *WAKE* =.FALSE. or *Initial*=.TRUE. which means):
- i.  $A(J,IBlade) = AP(J,IBlade) = 0.0$
  4. Calculate  $Vinduced = VNW * A(J,IBLADE)$  (from definition and BEM model)
  5. Calculate  $VN = VNW + VNB - Vinduced$
  6. Calculate  $SumInfl$  (*InducedVel*)=  $SumInfl + Vinduced * RLOCAL * DR(J)$  (*Blade*) (Note it was initialized before the loop on blade elements in *AD CalculateLoads (Aerodyn)*)
  7. Calculate Inflow Angle and AOA:
    - a.  $\text{PHI} = \text{ATAN2}(VN, VT)$  This would be the AOA if no local pitch (twist)
    - b.  $\text{ALPHA}(J,IBlade)$  (*ElemInflow*)=  $\text{PHI} - \text{PITNOW}$  (*Element*) actual angle of attack
  8. Put  $\text{ALPHA}$  between  $-\text{Pi}$  and  $\text{Pi}$  (Call *MPi2Pi(ALPHA(J,IBlade))* (*NWTC Num* within *NWTC Library* package)
  9. Calculate the square of the inflow velocity:  $W2(J,IBlade) = VN^{**2} + VT^{**2}$  (*ElemInflow*)
  10. Get Reynolds number for the blade element in millions:  $ReNum = \text{GetReynolds}(SQRT(W2(J,IBlade)), C(J))$  (*Blade*) )
  11. IF (*Reynolds*) (*Switch*) THEN *MulTabLoc (Airfoil)*= *ReNum*
  12. IF *DSTALL* (*Switch*) THEN: Now if *DSTALL* was activated by user in input, then calculate *Cl*, *Cd* via Beddoes. (Note that induction factors were calculated via static *Cl*, *Cd* though):
    - a. IF *Initial* THEN : this means it is the very first time step, use static data
      - i. Call *BEDINIT (J, IBlade, ALPHA(J,IBlade))*
      - ii. Call *CLCD (J, IBlade, ALPHA(J,IBlade))*
    - b. ELSE Call *BEDDOES( W2(J,IBlade), J, IBlade, ALPHA(J,IBlade), CLA, CDA, CMA)* (*AeroSubs*)
  13. ELSE *CLCD( ALPHA(J,IBlade), CLA, CDA, CMA, NFOIL(J), ErrStat )* static airfoil properties
  14. Calculate force base factor:  $QA = 0.5 * RHO * W2(J,IBlade) * DR(J) * C(J)$
  15. Calculate sin and cos of *PHI*: *CPHI*, *SPHI*
  16. Calculate the normal and tangential blade element force:
    - a.  $DFN = ( CLA * CPHI + CDA * SPHI ) * QA$
    - b.  $DFT = ( CLA * SPHI - CDA * CPHI ) * QA$
  17. IF *PMOMENT* (*Switch*) THEN *PMA*=*CMA*\* *QA* \**C(J)* ELSE *PMA* = *CMA*= 0.0
  18. IF *IBlade* ==1 .AND. *ElPrList(j)* (*ElOutParams*)> 0 THEN save aero info for the element; all from *ElOutParams*
    - a.  $AAA( ElPrList(J) ) = A(J,IBLADE) \& AAP( ElPrList(J) ) = AP(J,IBLADE)$
    - b.  $ALF( ElPrList(J) ) = ALPHA(J,IBlade) * R2D$  [deg]
    - c.  $CDD( ElPrList(J) ) = CDA \& CLL( ElPrList(J) ) = CLA \& CMM( ElPrList(J) ) = CMA$
    - d.  $CNN( ElPrList(J) ) = CLA * \text{COS}(ALPHA(J,IBlade)) + CDA * \text{SIN}(ALPHA(J,IBlade))$
    - e.  $CTT( ElPrList(J) ) = -CDA * \text{COS}(ALPHA(J,IBlade)) + CLA * \text{SIN}(ALPHA(J,IBlade))$
    - f.  $DFNSAV( ElPrList(J) ) = DFN \& DFTSAV( ElPrList(J) ) = DFT$
    - g.  $DynPres( ElPrList(J) ) = 0.5 * RHO * W2(J,IBlade)$
    - h.  $PITSBV( ElPrList(J) ) = PITNOW * R2D$

- i.  $PMM (ElPrList(J)) = PMA$
- j.  $ReyNum (ElPrList(J)) = ReNum$

## OUTPUT

*DFN, DFT, PMA:* R(ReKi). These are the blade element forces normal and tangent to rotational plane, and aero-torque. It also fills *A(Ielm, IBlade)* and *AP(Ielm, IBlade)* by calling the appropriate routines. It calculates *W2(Ielm, IBlade)* and *ALPHA(Ielm, IBlade)* (*ElemInflow*). It further stores arrays(*NumElOut*) from *ElOutParams*: *AAA, AAP, ALF, CDD, CLL, CMM, CNN, CTT, DFNSAV, DFTSAV, DynPres, PITSAV, PMM, ReyNum*. It assigns *MulTabLoc (Airfoil)* in case *Reynolds (Switch)* option is selected. It may also change *VT* total velocity in the plane of rotation.

## C22. VIND (J, IBLADE, RLOCAL, VNROTOR2, VNW, VNB, VT)

Subroutine that calculates the axial induction factor for each annular segment. It is called by *ElemFrc* (*AeroSubs*).

### THEORY

BEM and Generalized Dynamic Wake.

### EXTERNAL (INVOKED) MODULES

- *Blade* [see AeroMods.f90]
- *Element* [see AeroMods.f90]
- *InducedVel* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
RLOCAL	R(ReKi)	IN	Local radial distance of element
VNB	R(ReKi)	IN	Component of the translational velocity normal to plane of rotor
VNROTOR2	R(ReKi)	IN	square of the wind velocity normal to the rotor plane
VNW	R(ReKi)	IN	Component of wind velocity normal to rotational plane (rotor)
VT	R(ReKi)	INOUT	Tangential velocity accounting for wind and rotation/deflections of blade
J	I	IN	Blade element counter
IBlade	I	IN	Blade counter
Initial	L	IN	First pass or not at load calculation for that blade element

Other input/output variables based on invoked modules: e.g.: *A* and *AP(Element)* .

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>A2</i>	R(ReKi)			New value for A: induction factor
<i>A2P</i>	R(ReKi)			New value for AP: induction factor
<i>AI</i>	R(ReKi)			Initial value for iteration on A: induction factor
<i>ALPHA</i>	R(ReKi)			Aero AOA calculated accounting for induction factors and associated inflow angle
<i>ASTEP</i>	R(ReKi)			Relaxation step for Delta_A
<i>ATOLER2</i>	R(ReKi)			$2 * ATOLER$ ( <i>InducedVel</i> )

<i>ATOLERBY10</i>	R(ReKi)	0.1* <i>ATOLER (InducedVel)</i>
<i>CDA</i>	R(ReKi)	<i>Cd</i> calculated accounting for induction factors and associated inflow angle
<i>CLA</i>	R(ReKi)	<i>Cl</i> calculated accounting for induction factors and associated inflow angle
<i>CMA</i>	R(ReKi)	<i>Cm</i> calculated accounting for induction factors and associated inflow angle
<i>DAI</i>	R(ReKi)	<i>Delta_A</i> between current and last iterations
<i>DA1</i>	R(ReKi)	Previous iteration <i>Delta_A</i>
<i>DELAI</i>	R(ReKi)	Relaxed <i>Delta_A</i>
<i>PHI</i>	R(ReKi)	Inflow Angle calculated accounting for induction factors
<i>OLD_A_NS(:,::)</i>	R(ReKi)	SAVE Previous time step <i>A</i> factor before skew correction
<i>OLD_AP_NS(:,::)</i>	R(ReKi)	SAVE Previous time step <i>AP</i> factor before skew correction
<i>SOLFACT</i>	R(ReKi)	Solidity factor $NB^*C/(2*\pi*RLOCAL^*VNROTOR2)$
<i>VNA</i>	R(ReKi)	Effective normal to plane of rotation velocity
<i>VT2_Inv</i>	R(ReKi)	$=1/VT^2$
<i>VTA</i>	R(ReKi)	Effective tangent to plane of rotation velocity
<i>ICOUNT</i>	I(4)	Iteration counter
<i>MAXICOUNT</i>	I(4)	Maximum number of iterations allowed
<i>Sttus</i>	I	Error Status Flag

**STEPS**

1. IF NOT. Allocated ALLOCATE *OLD\_A\_NS*(*Nelm (Element)*, *NB(Blade)*) and *OLD\_AP\_NS*(*Nelm*, *NB*)
2. Set *MAXICOUNT* =1000, maximum number of iterations allowed. Should be put as parameter somewhere else, minor bug.
3. Set *ATOLER2* =2\**ATOLER* and *ATOLERBY10*=0.1\**ATOLER (InducedVel)*
4. IF velocity is less than 3 m/s: IF *VNROTOR2* < 0.1 THEN: set *A(J,IBlade)*=0.0 and RETURN
5. IF *RLOCAL* ==0.0 THEN *SOLFACT*=1.0/*VNROTOR2* to avoid div by 0 (solidity factor)
6. ELSE *SOLFACT*=*NB*\**C(J)(Blade)*/(2\*pi\*i*RLOCAL*\**VNROTOR2*)

7. Set  $VT2\_Inv = 1./ VT^2$
8. Set the following variables:
  - a.  $AI = OLD\_A\_NS(J, IBLADE)$  : previous time step value for  $A$  assigned to initial value  $AI$
  - b.  $DAI1 = 0.05$
  - c.  $A2P = OLD\_AP\_NS(J, IBLADE)$  : previous time step value for  $AP$  assigned to initial value  $A2P$
  - d.  $ASTEP = 0.5$
  - e.  $ICOUNT = 0$
9. Check if velocities are too high and bypass calcs in case: IF ( ABS(  $VNB$  ) > 100. ) THEN
  - a. Set  $A(J, IBLADE) = 0.0$  & Call VINDERR(  $VNW, VNB, 'VNB', J, IBLADE$  ) (*AeroSubs*)
  - b. RETURN
10. ELSEIF ABS(  $VT$  ) > 400. THEN
  - a. Set  $A(J, IBLADE) = 0.0$  & Call VINDERR(  $VNW, VT, 'VT', J, IBLADE$  ) (*AeroSubs*)
  - b. RETURN
11. Set  $A2 = AI$ ;  $A2$  and  $A2P$  are the new  $A$  values to be calculated:
12. 1<sup>st</sup> ITERATION: Call AXIND(  $VNW, VNB, VNA, VTA, VT, VT2\_Inv, VNROTOR2, A2, A2P, J, SOLFACT, ALPHA, PHI, CLA, CDA, CMA, RLOCAL$  )
13. Set delta\_A and relaxed version for delta\_A:  $DAI = A2 - AI$  and  $DELAI = ASTEP * DAI$
14. ITERATE TO FIND A AND AP:
 

DO\_LOOP WHILE  $|DELAI| > ATOLERBY10$ .AND.  $ABS(DAI) > ATOLER2$  (double convergence criteria to be satisfied; it helps when crossing zero many times)

  - a. Update Iteration counter:  $ICOUNT = ICOUNT + 1$
  - b.  $A2 = AI$
  - c. Call AXIND(  $VNW, VNB, VNA, VTA, VT, VT2\_Inv, VNROTOR2, A2, A2P, J, SOLFACT, ALPHA, PHI, CLA, CDA, CMA, RLOCAL$  )
  - d. Update delta\_A and relaxed delta\_A:  $DAI = A2 - AI$  and  $DELAI = ASTEP * DAI$
  - e. IF  $ICOUNT > MAXICOUNT$  THEN
    - i. warn user (ProgWarn NWTC\_IO)
    - ii. set  $A2 = OLD\_A\_NS(J, IBLADE)$  : use previous time step value
    - iii. set  $A2P = OLD\_AP\_NS(J, IBLADE)$  : use previous time step value
    - iv. EXIT DO\_LOOP
  - f. Reduce step size after 0-crossing:
    - i. IF(  $NINT(\text{SIGN}(1., DAI)) \neq NINT(\text{SIGN}(1., DAI1))$  ) THEN  $ASTEP = \max(1.0E-4, 0.5 * ASTEP)$  (note ASTEP not allowed to go below 1e-4)
  - g. SET  $AI = AI + DELAI$  and  $DAI1 = DELAI$
  - h. END\_LOOP
15. Out of the loop, convergence achieved:
  - a. Assign  $A(J, IBLADE) = A2$  and  $AP(J, IBLADE) = A2P$  (*Element*)
  - b.  $VT = VT * (1. + A2P)$  **NOT SURE WHY THIS IS HAPPENING**
  - c. Assign  $OLD\_A\_NS(J, IBLADE) = A2$  and  $OLD\_AP\_NS(J, IBLADE) = A2P$

## OUTPUT

$A(J,IBLADE)$   $AP(J,IBLADE)$  : R(ReKi). These are the blade element induction factors. It also fills  $OLD\_A\_NS$  ( $J,IBLADE$ ) and  $OLD\_A\_NS$  ( $J,IBLADE$ ) (R(ReKi)) that will be used the next time this routine is called, and they represent the previous time-step values for induction factors. The total velocity tangent  $VT$  (R(ReKi)) to the rotor plane is also updated for the induction effect (not sure why however).

## C23. VINDERR (VNW, VX, VID, J, IBLADE)

Subroutine that writes warning messages to the screen when *VNB* or *VT* are high. It is called by *VIND* (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

N/A

### INPUT

Name	Type	INTENT	Description
VNW	R(ReKi)	IN	Component of wind velocity normal to rotational plane (rotor)
VX	R(ReKi)	IN	Either <i>VNB</i> or <i>VT</i> (see <i>VIND AeroSubs</i> )
VID	C(*)	IN	String assuming either <i>VNB</i> or <i>VT</i> (see <i>VIND AeroSubs</i> )
J	I(4)	IN	Blade element counter
IBlade	I(4)	IN	Blade counter

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
NERRORS	I(4)	0	Initialized, SAVE	Counter of errors
AFLAG	L	.FALSE.	Initialized, SAVE	5-error message flag: after 5 error messages user is warned

### STEPS

1. IF *AFLAG* RETURN : this flag gets to .TRUE. after 5 message writing
2. Augment *NERRORS* = *NERRORS* +1
3. Call *ProgWarn* and *WrScr* (*NWTC\_IO* within *NWTC\_Library*) to warn user of high value encountered for velocity *VID* during induction calculation for a given *IBlade* and *IElm*, also showing the value of *VNW* and *VX* which is the value for *VID*
4. IF *NERRORS*>=5 set *AFLAG* = .TRUE. and warn user about that

### OUTPUT

It spits out messages on the screen if *VT* or *VNB* are too high (see *VIND AeroSubs*).

## C24. AXIND (VNW, VNB, VNA, VTA, VT, VT2\_INV, VNROTOR2, A2, A2P, J, SOLFACT, ALPHA, PHI, CLA, CDA, CMA, RLOCAL)

Subroutine that calculates a new axial induction factor from given values of velocities and geometry. It calculates a new value of  $A$  and  $AP$ , called  $A2$  and  $A2P$ . It is called by VIND (*AeroSubs*) as part of the iteration process.

### THEORY

See Harman's PROPX paper.

### EXTERNAL (INVOKED) MODULES

- *Element* [see AeroMods.f90]
- *InducedVel* [see AeroMods.f90]
  - *NFoil(:) (I(4)) only*
- *Airfoil* [see AeroMods.f90]
- *Switch* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
VNW	R(ReKi)	IN	Component of wind velocity normal to rotational plane (rotor)
VNB	R(ReKi)	IN	Component of the translational velocity normal to plane of rotor
VNA	R(ReKi)	OUT	Component of the total velocity normal to the plane of the rotor accounting for induction
VTA	R(ReKi)	OUT	Component of the total velocity tangent to the plane of the rotor accounting for induction
VT	R(ReKi)	IN	Total tangential velocity
VT2_Inv	R(ReKi)	IN	=1/VT <sup>2</sup> (VT=body+wind tangential vel. component)
VNROTOR2	R(ReKi)	IN	Square of the wind component normal to rotor plane
A2	R(ReKi)	INOUT	Axial induction factor
A2P	R(ReKi)	INOUT	Tangential induction factor
J	I(4)	IN	Blade element counter
SOLFACT	R(ReKi)	IN	Solidity factor $NB*C/(2*pi*RLOCAL*VNROTOR2)$
ALPHA	R(ReKi)	OUT	AOA
RLOCAL	R(ReKi)	IN	Radial distance to blade element in the plane of rotation
PHI	R(ReKi)	OUT	Inflow angle

<i>CLA</i>	R(ReKi)	OUT	Cl
<i>CDA</i>	R(ReKi)	OUT	Cd
<i>CMA</i>	R(ReKi)	OUT	Cm

**LOCAL VARIABLES**

Name	Type	Value	PARAMETER/ Initialized	Description
<i>CH</i>	R(ReKi)			BEM portion of the thrust equation (head loss)
<i>CPhi</i>	R(ReKi)			$\text{Cos}(\Phi)$
<i>HUBLOSS</i>	R(ReKi)	1	Initialized, SAVE	Hubloss factor
<i>LOSS</i>	R(ReKi)	1	Initialized, SAVE	$=TIPLOSS*HUBLOSS$
<i>SPHI</i>	R(ReKi)			$\text{Sin}(\Phi)$
<i>SWRLARG</i>	R(ReKi)			Argument for square root in AP calculation
<i>TIPLOSS</i>	R(ReKi)	1	Initialized, SAVE	Tiploss factor
<i>W2</i>	R(ReKi)			Square of the total relative velocity
<i>ErrStat</i>	I			Error Status Flag

**STEPS**

1. Set  $VNA = VNW * (1.-A2) + VNB$ ; effective axial velocity using previous value of  $A$ ; Note body motion added a-posteriori here, but not for VTA!
2. Set  $VTA = VT * (1. + A2P)$ ; effective tangential velocity using previous value of  $AP$
3. Calculate adjusted ALPHA accounting for induction:
  - a.  $\text{PHI} = \text{ATAN2}(VNA, VTA)$
  - b.  $\text{ALPHA} = \text{PHI} - \text{PITNOW}$  (*Element*) and put into  $-\text{Pi}, \text{Pi}$  (Call **MPI2Pi(ALPHA)** *NWTC\_Num* within *NWTC\_Library*)
4. Calculate Cl and Cd from static values: Call **CLCD(ALPHA, CLA, CDA, CMA, NFoil(J), ErrStat)** (*AeroSubs*)
5. Calculate:
  - a.  $W2 = VNA^2 + VTA^2$ , square of the relative air-velocity
  - b.  $SPHI = VNA/\text{SQRT}(W2)$ ,  $\sin(\text{PHI})$
  - c.  $CPhi = \text{COS}(\text{PHI})$
  - d.  $CH = W2 * \text{SOLFACT} * (\text{CLA} * CPhi + EqAIDmult(\text{InducedVel}) * \text{CDA} * SPhi)$  (LHS (BEM) of the Thrust Equation); Note the addition or not of the drag term in the thrust equation
6. IF *TLOSS* (*Switch*) THEN Call **GetTipLoss(J, SPhi, TIPLOSS, RLOCAL)**
7. IF *HLOSS* (*Switch*) THEN Call **GetPrandtlLoss(HLCNST(J)(Element), SPhi, HUBLOSS)**

8.  $LOSS = TIPLOSS * HUBLOSS$
9. IF  $ABS(CH) > 2$  THEN Limit  $CH=2 * SIGN(CH)$ ; this apparently fixes diverging problems
10. Get NEW VALUES of A(i.e. A2): Check the braked-windmill state, and in case use Glauert's correction:
  - a. IF ( $CH < 0.96 * LOSS$ ) THEN  $A2 = 0.5 * (1 - \sqrt{1.0 - CH/LOSS})$ : This is standard BEM
  - b. ELSE  $A2 = 0.1432 + \sqrt{-0.55106 + .6427 * CH/LOSS}$  Note this is not Marshal's correction, but more similar to the original Glauert's parabola for  $a > 0.5$
11. IF SWIRL (*Switch*) THEN: Calculate AP, tangential induction following Ross Harmon's PROP-PC
  - a. IF EquildT (*Switch*) THEN: use drag term in angular momentum equation
    - i. Also avoid problems near  $\Phi=0^\circ$  or  $90^\circ$ :
    - ii. IF  $ABS(SPhi) > 0.01$  AND  $ABS(CPhi) > 0.01$  THEN  $A2P = SOLFACT * (CLA * SPhi - CDA * CPhi) * (1.0 + A2P) * VNROTOR2 / (4.0 * LOSS * SPhi * CPhi)$ ; Note the VNROTOR2 is there since SOLFACT has that term at the denominator which is not in the torque equation for AP
    - iii. ELSEIF  $ABS(SPhi) > 0.01$ : this means  $\Phi$  approaching  $90^\circ$ : THEN in the A2P formula set  $CPhi = 0.01 * sign(CPhi)$
    - iv. ELSE this means  $\Phi$  approaching  $90^\circ$ : THEN in the A2P formula set  $SPhi = 0.01 * SIGN(SPhi)$
  - b. ELSE: no drag term in the tangential induction calculation using Goldstein's hypothesis:
    - i. Set the argument of a square root as  $SWRLARG = 1.0 + 4.0 * LOSS * A2 * VNW * VNA * VT2\_Inv$
    - ii. IF  $SWRLARG < 0.0$  THEN  $A2P = 0.0$  ELSE  $A2P = 0.5 * (-1.0 + \sqrt{SWRLARG})$ : this is from PROPX with the addition of  $VT^2$  at the denominator rather than just  $\Omega_r$  WHY NOT THE COMPLETE VN at the numerator?
12. ELSE: no SWIRL model, no tangential induction requested THEN  $A2P = 0.0$

## OUTPUT

$A2, A2P, PHI, CLA, CDA, CMA, ALPHA, VTA, VNA$  : real(ReKi). It calculates new values of A and AP called A2 and A2P. Also the components normal and tangential to rotor plane of total velocity VNA and VTA accounting for previous values of A and AP; thus, the inflow angle  $\Phi$  and Aero-AOA  $ALPHA$ , and the associated aero coefficients  $CLA, CDA, CMA$ . These are associated to the A2 and A2P of the previous iteration step, and are needed to calculate the new values A2 and A2P.

## C25. GETPRANDTLLOSS (LCNST, SPHI, PRLOSS)

Subroutine that calculates hub loss constant for blade element under investigations. It is called by *AXIND* (*AeroSubs*) as part of the iteration process. Also Called by *GetTipLoss* (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

N/A

### INPUT

Name	Type	INTENT	Description
<i>LCNST</i> R(ReKi)	IN		This is the argument $NB^*(R-r)/(2^*r)$ or $NB^*(r-RHUB)/(2^*RHUB)$ in Prandtl's exponential function
<i>SPHI</i> R(ReKi)	IN		Sin of Inflow angle Phi
<i>PrLOSS</i> R(ReKi)	OUT		Loss (F)

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/	Description
			Initialized	
<i>F</i>	R(ReKi)			Exponential function argument in Prandtl's model

### STEPS

1. Check if SPHI is too small to bother : IF  $|SPHI| < 1.E-4$  THEN  $PrLOSS=1.0$
2. ELSE:
  - a.  $F = ABS(LCnst / SPHI)$  is the exponent for  $e^()$
  - b. IF  $F < 7.$  THEN  $PrLOSS= ACOS( EXP( -F ) ) / (\Pi/2)$
  - c. ELSE  $PrLOSS = 1.0$ , avoids underflow errors when exponent is way small

### OUTPUT

*PrLOSS* : real(ReKi). It calculates HUBLOSS via Prandtl's model.

## C26. GETTIPLOSS (J, SPHI,TIPLOSS, RLOCAL)

Subroutine that calculates tip loss constant for blade element under investigations. Uses the Prandtl tip loss model with a correction from Georgia Tech (2002 ASME Wind Energy Symposium). It is called by *AXIND* (*AeroSubs*) as part of the iteration process.

### EXTERNAL (INVOKED) MODULES

- *Element* [see AeroMods.f90]
- *Blade*[see AeroMods.f90]
- *Switch* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
<i>J</i>	I(4)	IN	This is the argument $NB^*(R-r)/(2^*r)$ or $NB^*(r-RHUB)/(2^*RHUB)$ in Prandtl's exponential function
<i>SPHI</i>	R(ReKi)	IN	Sin of Inflow angle Phi
<i>RLOCAL</i>	R(ReKi)	IN	Radial distance to blade element (this is actually in the plane of rotation, <b>but it is not correct if coning exists, it should be along pitch axis instead</b> )
<i>TIPLOSS</i>	R(ReKi)	OUT	Loss (aka F in the theory)

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>Dist2pt7</i>	R(ReKi)	0.7	Initialized	Current element distance to r/R = 0.7
<i>OLDDist7</i>	R(ReKi)			Previous element distance to r/R = 0.7
<i>percentR</i>	R(ReKi)			Blade element r/R
<i>TLpt7</i>	R(ReKi)		SAVE	Tiploss factor at r/R = 0.7
<i>Jpt7</i>	I(4)	0	Initialized	The element closest to r/R = 0.7
<i>FirstPass</i>	L	.TRUE.	Initialized, SAVE	Whether or not it is the first time the blade is scanned for the r/R=0.7 tip-loss

### STEPS

1. Start with Prandtl's Model: Call *GetPrandtlLoss*(TLCNST(*J*) (*Element* and initialized in *AD\_Init Aerodyn*, *SPHI*, *TIPLOSS*)
2. IF GTECH (*Switch* and initialized in *AD\_GetInput AeroSubs*) THEN apply Georgia Tech Model:
  - a. Set *percentR=RLOCAL/R* (*Blade*)
  - b. IF *FirstPass* THEN search for blade element closest to r/R=0.7:

- i. Compare current and previous: IF  $\text{ABS}(percentR - 0.7) < Dist2pt7$  THEN
  - 1.  $OLDDist7 = Dist2pt7$  put current into old value
  - 2.  $Dist2pt7 = \text{ABS}(percentR - 0.7)$  set this as new current value for distance to r/R=0.7 This is not SAVED or reused so why needed? POSSIBLE BUG
  - 3.  $Jpt7=J$  update the element index to current
  - 4.  $TLpt7=TIPLOSS$  assign to this element the calculated Prandtl's loss
- ii. IF  $J == NELM (\text{Element})$  THEN: Check if at the end of the blade:
  - 1. Set  $FirstPass = .FALSE.$
- iii. ELSE:
  - 1. RETURN: do not do correction till the correct  $TLpt7$  is calculated
- c. IF  $J==Jpt7$  THEN  $TLpt7 = TIPLOSS$
- d. IF  $percentR >= 0.7$  THEN apply actual correction:
  - i.  $TIPLOSS = (TIPLOSS**0.85 + 0.5) / 2.0$
  - e. ELSE  $TIPLOSS = 1.0 - percentR * (1.0 - TLpt7) / 0.7$

## OUTPUT

*TipLOSS:* R(ReKi). It calculates Tip-loss via Prandtl's model and, in case requested, applying Georgia Tech correction.

## C27. CLCD (ALPHA, CLA,CDA, CMA, I, ErrStat)

Subroutine that returns values of lift and drag coefficients. It interpolates airfoil coefficients from one or more tables (Reynolds) of airfoil data. Called by: *AXIND*, *ELEMFRC* (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

- *Airfoil* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
<i>ALPHA</i>	R(ReKi)	INOUT	Aero AOA
<i>CLA</i>	R(ReKi)	OUT	CL for requested ALPHA
<i>CDA</i>	R(ReKi)	OUT	CD for requested ALPHA
<i>CMA</i>	R(ReKi)	OUT	CM for requested ALPHA
<i>I</i>	I(4)	IN	<i>NFOIL(J)(Airfoil)</i> , i.e. the airfoil file # for the current blade element
<i>ErrStat</i>	I	OUT	>0 means error encountered

Additional Input via invoked module variables.

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>CDA1</i>	R(ReKi)			Auxiliary Cd value due to interpolation along ALPHA direction
<i>CDA2</i>	R(ReKi)			Auxiliary Cd value due to interpolation along Reynolds Number direction
<i>CLA1</i>	R(ReKi)			Auxiliary Cl value due to interpolation along ALPHA direction
<i>CLA2</i>	R(ReKi)			Auxiliary Cl value due to interpolation along Reynolds Number direction
<i>CMA1</i>	R(ReKi)			Auxiliary Cm value due to interpolation along ALPHA direction
<i>CMA2</i>	R(ReKi)			Auxiliary Cm value due to interpolation along Reynolds Number direction
<i>P1</i>	I(4)			Auxiliary linear interpolation ratio along ALPHA

<i>P2</i>	<i>I(4)</i>	Auxiliary linear interpolation ratio along Reynolds Numbers
<i>N1</i>	<i>I(4)</i>	Auxiliary index used in interpolation (lower value index)
<i>N1P1</i>	<i>I(4)</i>	<i>N1+1</i>
<i>N2</i>	<i>I(4)</i>	Auxiliary index used in interpolation (lower value index)
<i>N2P1</i>	<i>I(4)</i>	<i>N2 +1</i>
<i>NTAB</i>	<i>I(4)</i>	Auxiliary=number of aerodata rows in file

**STEPS**

1. IF .NOT. ALLOCATED(*NFoil*) (*Airfoil*) : i.e. if no list of airfoil data file exists THEN
  - a. *CDA* = *CLA* = *CMA*
  - b. *ErrStat*=1
  - c. RETURN
2. ELSE *ErrStat*=0
3. *NTAB*= *NLIFT(I)* (*Airfoil*) number of Aerodata points in the current file
4. IF *ALPHA* < *AL(I,1)* (*Airfoil*) .OR. ( *ALPHA* > *AL(I,NTAB)* ) : out-of-bounds THEN
  - a. Warn user and abort Call *ProgAbort()* (*NWTC\_IO* within *NWTC\_Library*)
  - b. *ErrStat*=1
  - c. Note, however, that the following interpolation would allow for out-of-bound values of *ALPHA*
5. Start the linear interpolation by assigning in a clever way the AOA to be the table upper or lower bound if *ALPHA*= MIN( MAX( *ALPHA*, *AL(I,1)* ), *AL(I,NTAB)* )
6. Find the closest lower value and index *N1* associated with the value *ALPHA* within *AL(I,:)* (Call *LocateBin NWTC\_Num* within *NWTC\_Library*)
7. IF *N1* ==0 .OR. *N1* ==*NTAB* : Adjust the values of *N1*, *N1P1* to remain within data table, and also assign *P1*: (minor bug: this can be written in a cleverer way since *N1P1* always =*N1+1*.)
8. ELSE *P1*= (*ALPHA* - *AL(I, N1)* ) / ( *AL(I, N1P1)* - *AL(I, N1)* ) to be used in interpolation
9. IF *NTABLES(I)* (*Airfoil*)> 1 THEN: multiple tables (Reynolds numbers) exist for the current file
  - a. Analogously to *ALPHA* Set *MulTabLoc* (*Airfoil*)= MIN( MAX( *MulTabLoc*, *MulTabMet(I,1)* (*Airfoil*)), *MulTabMet(I,NTables(I))* ) so that it stays within *MulTabMet(I,:)* (e.g.:Reynolds) number bounds for instance
  - b. Find the closest lower value and index *N2* associated with the value *MulTabLoc* within *MulTabMet(I,:)* (Call *LocateBin NWTC\_Num* within *NWTC\_Library*)
  - c. IF *N2* ==0 .OR. *N2* ==*NTables(I)* : Adjust the values of *N2*, *N2P1* to remain within data table range, and also assign *P2*: (minor bug: this can be written in a cleverer way since *N2P1* always =*N2+1*.)
  - d. ELSE *P2* = (*MulTabLoc* - *MulTabMet(I,N2)*)/(*MulTabMet(I,N2P1)-MulTabMet(I,N2)*)

- e. Calculate  $CLA1$   $CDA1$   $CMA1$  and  $CLA2$   $CDA2$   $CMA2$  as linear interpolations along  $ALPHA$  direction of the values at  $MulTabMet(I,N2)$  and  $MulTabMet(I,N2P1)$ :
    - i.  $C[L/M/D]A1 = C[L/M/D](I,N1,N2) + P1 * ( C[L/M/D] (I,N1P1,N2) - C[L/M/D] (I,N1,N2) )$
  - f. Finally Calculate  $CLA$   $CDA$   $CMA$  as linear interpolations along  $MulTabMet$  of the  $CLA1..CMA1$  and  $CLA2..CMA2$  :
    - i.  $C[L/M/D]A = C[L/M/D]A1 + P2 * ( C[L/M/D]A2 - C[L/M/D]A1 )$
10. ELSE only 1 Table in the file so simple linear interpolation along  $ALPHA$  direction:
- i.  $C[L/M/D]A = C[L/M/D](I,N1,1) + P1 * ( C[L/M/D] (I,N1P1,1) - C[L/M/D] (I,N1,1) )$

## OUTPUT

$CLA$ ,  $CDA$ ,  $CMA$  :  $\text{real}(\text{ReKi})$ . Static  $Cl$ ,  $Cd$ ,  $Cm$  bi-linearly interpolated from airfoil data file along  $ALPHA$  (AOA) and  $MulTabMet$  (e.g.: Reynolds Numbers).

## C28. VNMOD(J,IBLADE,RLOCAL,PSI)

Subroutine that modifies the axial induction factor for yaw/tilt/skewed-wake effects. Called by ELEMFRC (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

- *Blade* [see AeroMods.f90]
- *Element* [see AeroMods.f90]
- *Wind* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
<i>J</i>	I	IN	Blade element index
<i>IBLADE</i>	I	IN	Blade index
<i>RLOCAL</i>	R(ReKi)	IN	Radial distance of blade element in the plane of rotation
<i>PSI</i>	R(ReKi)	IN	Azimuth of blade under analysis

Additional input via variables in the invoked modules (e.g., *R* (*Blade*))

### LOCAL VARIABLES

A few auxiliary variables needed as counters in DO\_LOOP, or format strings.

Name	Type	Value	PARAMETER/ Initialized	Description
<i>BB</i>	R(ReKi)			$15\pi/64 * \text{SQRT}((1. - SANG)/(1. + SANG))$
<i>SANG</i>	R(ReKi)			$\text{Sin}(ANGFLW)$ ( <i>Wind</i> )

### STEPS

1. Set  $SANG = \text{SIN}(ANGFLW)$  (*Wind*);  $ANGFLW$  calculated in DiskVel (*AeroSubs*);
2. Calculate  $BB = 0.7363 * \text{SQRT}((1. - SANG)/(1. + SANG))$ ; note  $0.7363 = 15\pi/64$
3. Modify the axial induction factor as follows:
4.  $A(J,IBLADE) = A(J,IBLADE) * (1. + 2. * RLOCAL/R(Blade) * BB * (SDEL(Wind) * \text{SIN}(PSI) + CDEL(Wind) * \text{COS}(PSI)))$ ; note  $SDEL$  and  $CDEL$  calculated in DiskVel (*AeroSubs*);

### OUTPUT

$A(J,iblade)$ : R(ReKi) (*Element* in *AeroMods*): it modifies the axial induction factor to account for off-axis flow (tilt and yaw)

## C29. SAT(X, VAL, SLOPE)

Function that applies a saturation to the AOA. Called by ATTACH (AeroSubs).

### THEORY

It modifies the input value if above a certain threshold, else it leaves it as is. Above the threshold, it uses a linear function passing through the threshold and with a given slope (less than 1). The y value associated with that line and the x input is the saturated output.

### EXTERNAL (INVOKED) MODULES

N/A

### INPUT

Name	Type	INTENT	Description
X	R(ReKi)	IN	Value to be compared against saturation
VAL	R(ReKi)	IN	Saturation Threshold
SLOPE	R(ReKi)	IN	Slope to use in saturation trimming

### LOCAL VARIABLES

N/A

### STEPS

1. IF ABS(X) <= VAL THEN
  - a.  $SAT = X$ ; input is below threshold, do nothing
2. ELSEIF X > VAL THEN
  - a.  $SAT = SLOPE * X + VAL * (1. - SLOPE)$
3. ELSE: this means X < -VAL
  - a.  $SAT = SLOPE * X + VAL * (1. - SLOPE)$

### OUTPUT

SAT: R(ReKi): it modifies the axial induction factor to account for off-axis flow (tilt and yaw)

## C30. INFLOW()

Subroutine that leads to the dynamic inflow routines. Called by *AD CalculateLoads* (*Aerodyn*).

### EXTERNAL (INVOKED) MODULES

- *AeroTime* [see AeroMods.f90]
- *DynInflow* [see AeroMods.f90]
- *Switch* [see AeroMods.f90]

### INPUT

Input is via variables within invoked modules.

### LOCAL VARIABLES

N/A

### STEPS

1. IF *DYNINFL* (*Switch*) THEN:
  - a. IF *DYNINIT* (*Switch*) AND *TIME*>0.0 (*Aerodyn*) THEN
    - i. Call *INFINIT* (*AeroSubs*)
    - ii. *Old\_Alph*= 0.0 (*DynInflow*)
    - iii. *Old\_Beta* = 0.0 (*DynInflow*)
    - iv. *DYNINIT* = .FALSE. (*Switch*)
    - v. *SKEW* = .FALSE. (*Switch*)
  - b. Call *INFUPDT* (*AeroSubs*) which updates GDW variables
  - c. Call *GetPhiLq* (*AeroSubs*) which accumulates the rotor forces for dynamic inflow calculations
  - d. IF( *TIME* (*AeroTime*) > 1.0D0 ) Call *INFDIST* (*AeroSubs*) which calculates the inflow (induced flow) distribution parameters using GDW Theory.

### OUTPUT

It initializes *Old\_Alph* and *Old\_Beta* (*DynInflow*); *DYNINIT* and *SKEW* (*Switch*). By calling other subroutines it also calculates/updates many GDW variables.

### C31. GETRM (RLOCAL, DFN, DFT, PSI,J, IBLADE)

Subroutine that returns the mode-moments of the blade elemental force, i.e: the blade elemental force is multiplied by the radial shape function for all of the modes. Also see function XPHI. Called by: AD CalculateLoads, (*AeroSubs*).

#### EXTERNAL (INVOKED) MODULES

- *Blade* [see AeroMods.f90]
- *DynInflow* [see AeroMods.f90]
- *Switch* [see AeroMods.f90]

#### INPUT

Name	Type	INTENT	Description
<i>rLocal</i>	R(ReKi)	IN	Blade element radial distance
<i>DFN</i>	R(ReKi)	IN	Normal force on element
<i>DFT</i>	R(ReKi)	IN	Tangential force on element
<i>psi</i>	R(ReKi)	IN	Blade Azimuth
<i>J</i>	I	IN	Blade Element Index
<i>IBlade</i>	I	IN	Blade Index

Additional Input via invoked module variables (e.g., *R (Blade)*).

#### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>fElem</i>	R(ReKi)			Elemental force
<i>Rzero</i>	R(ReKi)			<i>rLocal/R (Blade)</i>
<i>WindPsi</i>	R(ReKi)			New azimuth
<i>mode</i>	I(4)			Counter for modes

#### STEPS

1. IF SWIRL (*Switch*) THEN:(i.e. calculate tangential induction as well)
  - a.  $fElem = \sqrt{DFN^2 + DFT^2}$  ; vector sum of tangential and normal forces
2. ELSE  $fElem = DFN$  ; only normal force
3. Set  $fElem = fElem / R$  (*Blade*) : Note this is done to calculate  $\tau_n^{mc}$  and  $\tau_n^{ms}$  correctly, see InflInit and InfDist (*AeroSubs*)
4. Set  $Rzero=rLocal / R$  ; non-dimensional radial coordinate
5. Now set a new azimuth, based on FF<sub>HR</sub>: Call WindAzimuthZero(*psi,WindPsi*) (*AeroSubs*)
6. DO\_LOOP *mode*=1, MAXINFL0 (*DynInflow*)

- a.  $RMC\_SAVE(IBLADE, J, mode) = fElem * \underline{XPHI}(Rzero, mode)$  (AeroSubs); this is for m=0 harmonic, which has 2 radial shape functions and thus 2 states associated with it. Since it has been chosen to limit the power of Rzero to 3 (see XPHI (AeroSubs) and DynInflow).
- 7. END\_LOOP
- 8. DO\_LOOP mode = MAXINFL0+1, maxInfl
  - a.  $RMC\_SAVE(IBLADE, J, mode) = fElem * \underline{XPHI}(Rzero, mode) * \text{COS}(\text{REAL}(\text{MRvector}(mode))$  (DynInflow) \* Windpsi ); Note MRvector is set in InflInit (AeroSubs)
  - b.  $RMS\_SAVE(IBLADE, J, mode) = fElem * \underline{XPHI}(Rzero, mode) * \text{SIN}(\text{REAL}(\text{MRvector}(mode))$  \* Windpsi )
- 9. END\_LOOP

## OUTPUT

It sets  $RMC\_SAVE(IBLADE, J, mode)$ ,  $RMS\_SAVE(IBLADE, J, mode)$  : real(ReKi), (DynInflow).

## C32. INFINIT ()

Subroutine that initializes variables in the GDW. Called by: Inflow (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

- *Blade* [see AeroMods.f90]
- *DynInflow* [see AeroMods.f90]
- *Element* [see AeroMods.f90]
- *Rotor* [see AeroMods.f90]
- *Wind* [see AeroMods.f90]
- *AeroTime* [see AeroMods.f90]

### INPUT

Additional Input via invoked module variables (e.g., *R* (*Blade*) *REVS* (*Rotor*), *RHO* (*Wind*)).

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>tauCos</i> ( <i>maxInfl</i> )	R(ReKi)			This is $\tau_n^{mc}$
<i>tauSin</i> ( <i>maxInfl0+1</i> : <i>maxInfl</i> )	R(ReKi)			This is $\tau_n^{ms}$
<i>V1</i>	R(ReKi)			Normalized Wind velocity component
<i>V2</i>	R(ReKi)			Normalized Wind velocity component
<i>V3</i>	R(ReKi)			Normalized Wind velocity component
<i>Vplane2</i>	R(ReKi)			Square of the in-plane wind velocity magnitude
<i>i</i>	I(4)			Loop Counter
<i>iElem</i>	I(4)			Counter for BEs
<i>irow</i>	I(4)			Counter for rows
<i>jBlade</i>	I(4)			Counter for blades
<i>jcol</i>	I(4)			Counter for columns
<i>k</i>	I(4)			Loop counter
<i>mode</i>	I(4)			Loop counter

### STEPS

1. Set the following arrays belonging to *DynInflow*: (Note this could be done elsewhere and defined as PARAMETERS minor bug)

- a.  $MRVector(:) = (/ 0, 0, 1, 1, 2, 3 /)$ ; indices of the azimuthal modes considered so far in AeroDyn, one per state
  - b.  $NJVector(:) = (/ 1, 3, 2, 4, 3, 4 /)$ ; indices of the azimuthal modes considered so far in AeroDyn, one per state and corresponding to the  $MRVector$  values
2. Initialize the time-derivative of the states:
- a.  $dAlpha_dt(:, :) = 0$ . (*DynInflow*)
  - b.  $dBeta_dt(:, :) = 0$ . (*DynInflow*)
3. Calculate and set the inverse of [M], which is a diagonal matrix, and thus stored as an array (see Theory Sections):
- a. DO\_LOOP  $irow = 1, maxInfl$ 
    - i.  $xMinv(irow) = PIBY2 (NWTC_Num \text{ in NWTC_Library}) / \underline{HFUNC}(MRvector(irow), NJvector(irow))$  (*AeroSubs*)
  - b. END\_LOOP
4. Now calculate the  $\Gamma$  function values used in the calculations of the [L] matrices, see also *FGAMMA* (*AeroSubs*) and put them in a matrix called *gamma*:
- a. DO\_LOOP  $irow = 1, maxInfl$ 
    - i. DO\_LOOP  $jcol = 1, maxInfl$ 
      1.  $gamma( irow, jcol ) = \underline{FGAMMA}( MRvector( irow ), NJvector( irow ), MRvector( jcol ), NJvector( jcol ) )$  (this could be calculated with *MRVector* somewhere else, minor bug)
    - ii. END\_LOOP
  - b. END\_LOOP
5. Now calculate the following matrices which are used in subroutine *LMATRIX* (*AeroSubs*) (this too could be calculated with *MRVector* somewhere else, minor bug)
- a. DO\_LOOP  $irow = 1, maxInfl$ 
    - i. DO\_LOOP  $jcol = 1, maxInfl$ 
      1.  $MminR( irow, jcol )$  (*DynInflow*) =  $\text{MIN}( MRvector(jcol) , MRvector(irow) )$
      2.  $MplusR( irow, jcol )$  (*DynInflow*) =  $MRvector(jcol) + MRvector(irow)$
      3.  $MminusR( irow, jcol )$  (*DynInflow*) =  $\text{ABS}( MRvector(jcol) - MRvector(irow) )$
    - ii. END\_LOOP
  - b. END\_LOOP
6. Now Calculate the tip speed of the rotor
- a.  $TipSpeed$  (*DynInflow*) =  $\text{MAX}(R(Blade) * REVS (Rotor), 1.0e-6)$
7. Now calculate the normalization factor:  $Pzero$  (*DynInflow*) =  $\pi * rho$  (*Wind*) \*  $TipSpeed^{**2} * R$  (This is correct, see Theory sections )
8. The non dimensional time interval is:  $DT0$  (*DynInflow*) =  $DT$  (*AeroTime*) \*  $REVS$
9. Set the wind velocities in  $FF_{HR}$  as normalized by the *TipSpeed*: Note the treatment here is non-dimensional, if RPM changes from time-step to time-step there may be problems integrating *GDW*
- a.  $v1 = VrotorZ (Wind) / TipSpeed$  ;inplane, upward

- b.  $v2 = VrotorY(Wind) / TipSpeed$ ; inplane, right looking downwind IT SHOULD BE LEFT!  
BUG?
  - c.  $v3 = -VrotorX(Wind) / TipSpeed$ ; normal to the rotor, pointing upwind
10. Calculate the initial value of the induced velocity  $\lambda_m$ : note that this is a spatially averaged value and it is calculated from the induction factors  $a$  which are calculated via BEMT at t=0.
- a. Initialize:  $xLambda_M = 0$ . (*DynInflow*)
  - b. DO\_LOOP  $jBlade=1,Nb$  (*Blade*)
    - i. DO  $iElem = 1,Nelm$  (*Element*)
      - 1.  $xLambda_M(DynInflow) = xLambda_M + A(iElem,jBlade)$  (*Element*)
      - ii. END\_LOOP
    - c. END\_LOOP
    - d.  $xLambda_M = xLambda_M / (Nb * Nelm)$ ; this is the spatially averaged  $a$
    - e.  $xLambda_M = xLambda_M * v3$ ; this is the actual  $\lambda_m$ , i.e. the normalized induction velocity normal to the rotor plane, here set positive if v3 is positive (unlikely, recommend revisiting signs as per Peters (2011))
11. The total wind velocity normal to the rotor plane is then: It seems like signs could be better taken bug?
- a.  $totalInf(DynInflow) = -v3 + xLambda_M$
12. The square of the inplane velocity (normalized) is (i.e., the square of the advance ratio):
- a.  $Vplane2 = v1 ** 2 + v2 ** 2$
13. The TOTAL WIND (INDUCTION INCLUDED) at the rotor plane is then:
- a.  $Vtotal(DynInflow) = \text{SQRT}(totalInf ** 2 + Vplane2)$
14. The mass parameter  $V_M$  is:
- a.  $Vparam(DynInflow) = (Vplane2 + (totalInf + old_LmdM) * totalInf) / Vtotal$  Why not current Lambda here? This seems incorrect bug?
15. The wake skew parameter is  $X_w$ :
- a.  $xkai(DynInflow) = \text{TAN}(.5 * \text{ATAN}(\text{SQRT}(Vplane2) / totalInf))$
16. Get the [L] matrices (See Theory):
- a. Call *LMATRIX(xKai, 1)* (*AeroSubs*); these are the L\_cos matrix
  - b. Call *LMATRIX(xKai, 2)*; these are the L\_sin matrix
17. Now calculate the  $\tau_n^{mc}$   $\tau_n^{ms}$ : start with the m=0 mode for cosines
- a. DO\_LOOP mode = 1, maxInfl0 (*DynInflow*)
    - i.  $\tau_{\text{cos}}(\text{mode}) = -\text{PhiLqC}(\text{mode})(DynInflow) / Pzero * .5$
    - ii. END\_LOOP ; NOTE at initial time step  $\text{PhiLqC}$  is calculated through results of BEMT
  - b. DO\_LOOP mode = maxInfl0 +1, maxInfl (*DynInflow*); the other terms both cos and sin for m>0
    - i.  $\tau_{\text{cos}}(\text{mode}) = -\text{PhiLqC}(\text{mode}) / Pzero$
    - ii.  $\tau_{\text{sin}}(\text{mode}) = -\text{PhiLqS}(\text{mode})(DynInflow) / Pzero$  ; NOTE at initial time step  $\text{PhiLqS}$  is calculated through results of BEMT

- c. END\_LOOP
18. Now calculate the alpha and beta states at 0-th timestep : $[L]\{\tau\}/2 / [V]$ ; this assumes the time-derivative is 0 at  $t=0$ , otherwise one could calculate it as  $([V] [L]^{-1} + [M]/DT0)^{-1}\{\tau\}/2$  perhaps to be modified;
- Initialize:  $xAlpha(1)$  (*DynInflow*) = 0.
  - DO\_LOOP  $k = 1, maxInfl$ 
    - $xAlpha(1) = xAlpha(1) + xLcos(1,k)$  (*DynInflow*) $^*$   $\tau_{cos}(k)$  ; this is for  $m=0$ ; note  $xLcos$  calculated by L MATRIX (*AeroSubs*)
  - END\_LOOP
  - $xAlpha(1) = .5 * xAlpha(1) / Vtotal$
  - now look at the other states for  $m=0, n > 1$ :
    - DO\_LOOP  $i = 2, maxInfl0$ 
      - Initialize:  $xAlpha(i) = 0$ .
      - DO\_LOOP  $k = 1, maxInfl$ 
        - $xAlpha(i) = xAlpha(i) + xLcos(i,k) * \tau_{cos}(k)$
      - END\_LOOP
      - $xAlpha(i) = .5 * xAlpha(i) / Vparam$
    - END\_LOOP
  - Now look at other states  $m > 0$ :
    - DO\_LOOP  $i = maxInfl0+1, maxInfl$ 
      - Initialize:  $xAlpha(i) = 0$ .
      - DO\_LOOP  $k = maxInfl0+1, maxInfl$ 
        - $xAlpha(i) = xAlpha(i) + xLcos(i,k) * \tau_{cos}(k)$
      - END\_LOOP
      - $xAlpha(i) = .5 * xAlpha(i) / Vparam$
      - Initialize:  $xBeta(i)$  (*DynInflow*) = 0.
      - DO\_LOOP  $k = maxInfl0+1, maxInfl$ 
        - $xBeta(i) = xBeta(i) + xLsin(i,k) * \tau_{sin}(k)$
      - END\_LOOP
      - $xBeta(i) = .5 * xBeta(i) / Vparam$
    - END\_LOOP
- Now invert the [L\_Cos] and [L\_Sin] matrices:
  - Call MATINV ( $xLcos, xLsin, maxInfl, maxInfl0, 1$ ) (*AeroSubs*);  $L_{cos}^{-1}$
  - Call MATINV ( $xLcos, xLsin, maxInfl, maxInfl0, 2$ ) (*AeroSubs*);  $L_{sin}^{-1}$

## OUTPUT

It sets or initializes the following arrays and variables in *DynInflow*:

*MRVector*(:) = (/ 0, 0, 1, 1, 2, 3 /); these indices are either m or r (k)

*NJVector*(:) = (/ 1, 3, 2, 4, 3, 4 /); these indices are for either n or j (l)

$dAlpha_dt(1:maxInfl,1:4) = 0.$  ;  $dBeta_dt(maxInfl0+1:maxInfl,1:4) = 0.$  : R(ReKi) ; the time derivatives of the states.

It then sets the following variables:

$MminR (maxInfl, maxInfl)$ ,  $MplusR (maxInfl, maxInfl)$ ,  $MminusR (maxInfl, maxInfl)$  : I(4); matrices built with the combinations of indices available for the states chosen, that will be used for the [L] matrix-build-up

$xMinv(1:maxInfl)$ : R(ReKi); the inverse matrix  $[M]^{-1}$

It calculates a matrix  $\gamma (maxInfl, maxInfl)$ : R(ReKi), made with  $\Gamma_{ln}^{km}$  function values

$DT0$ : R(ReKi); non-dimensional time,  $DT^*\Omega$

$Pzero$ : R(ReKi); normalization factor

$TipSpeed$ : R(ReKi); tip-speed updated to the current time step (0-th)

$xLambda_M$ : R(ReKi); normalized, induced velocity normal to rotor plane

$totalInfl$ : R(ReKi); total wind velocity normal to rotor plane (induction included)

$Vtotal$  : R(ReKi); total wind velocity magnitude at the rotor (induction included)

$Vparam$  : R(ReKi); the mass-flow parameter

$xkai$  : R(ReKi); the wake-skew parameter

$xAlpha(maxInfl)$  : R(ReKi); the alpha states at 0-th time step

$xBeta(maxInfl0+1:maxInfl)$  : R(ReKi); the alpha states at 0-th time step

It also sets the [L\_cos] and [L\_sin] matrices calling **LMATRIX (AeroSubs)**.

## C33. INFDIST()

Subroutine that calculates the inflow (induced flow) distribution parameters using GDW. Called by:  
Inflow (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

- *Blade* [see AeroMods.f90]
- *DynInflow* [see AeroMods.f90]
- *Element* [see AeroMods.f90]
- *Rotor* [see AeroMods.f90]
- *Wind* [see AeroMods.f90]
- *AeroTime* [see AeroMods.f90]

### INPUT

Additional Input via invoked module variables (e.g., *R* (*Blade*) *REVS* (*Rotor*), *RHO* (*Wind*)).

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>RHScos (maxInfl)</i>	R(ReKi)			RHS of GDW model equations (cosine terms)
<i>RHSSin(maxInfl0+1:maxInfl)</i>	R(ReKi)			RHS of GDW model equations (sine terms)
<i>tauCos (maxInfl)</i>	I(4)			This is $\tau_n^{mc}$
<i>tauSin(maxInfl0+1:maxInfl)</i>	I(4)			This is $\tau_n^{ms}$
<i>V1</i>	R(ReKi)			Normalized Wind velocity component
<i>V2</i>	R(ReKi)			Normalized Wind velocity component
<i>V3</i>	R(ReKi)			Normalized Wind velocity component
<i>Vplane2</i>	R(ReKi)			Square of the in-plane wind velocity magnitude
<i>i</i>	I(4)			Loop Counter
<i>k</i>	I(4)			Loop counter
<i>mode</i>	I(4)			Loop counter

### STEPS

1. Calculate the tip speed of the rotor at the current time step: Note the treatment here is non-dimensional, if RPM changes from time-step to time-step there may be problems integrating GDW (see Theory Sections)
  - a. *TipSpeed (DynInflow)= MAX(R(Blade) \* REVS (Rotor), 1.0e-6)*

2. Now calculate the normalization factor:  $Pzero = \pi * rho (\text{Wind})^* TipSpeed ^* 2 * R$  (This is correct since the extra R is included in the  $\Phi Lq[C/S]$ , see Theory sections)
3. The non dimensional time interval is:  $DT0 (\text{DynInflow})= DT (\text{AeroTime})^* REVS$
4. Set the wind velocities in FF<sub>HR</sub> as normalized by the *TipSpeed*:
  - a.  $v1 = VrotorZ (\text{Wind})/ TipSpeed$  ;inplane, upward
  - b.  $v2 = VrotorY (\text{Wind})/ TipSpeed$  ;inplane, right looking downwind **IT SHOULD BE LEFT!**  
**BUG?**
  - c.  $v3 = - VrotorX(\text{Wind}) / TipSpeed$  ; normal to the rotor, pointing upwind; Vrotor[X/Y/Z] calculated by DiskVel (*AeroSubs*)
5. The square of the inplane velocity (normalized) is:
  - a.  $Vplane2 = v1 ^* 2 + v2 ^* 2$
6. The total wind velocity normal to the rotor plane is then: **It seems like signs could be better taken bug?**
  - a.  $totalInf (\text{DynInflow})= - v3 + old\_LmdM(\text{DynInflow})$  ; it is defined positive downwind, while  $xLambda\_M$  is positive downwind(opposite to A); **Why not current Lambda here?**  
**This seems incorrect bug?**
7. The TOTAL WIND (INDUCTION INCLUDED) at the rotor plane is then:
  - a.  $Vtotal (\text{DynInflow})= \text{SQRT}( totalInf ^* 2 + Vplane2 )$
  - b. IF  $Vtotal <= 1.0e-6$  THEN  $Vtotal = 1.0e-6$
8. The mass parameter  $V_M$  is:
  - a.  $Vparam (\text{DynInflow})=( Vplane2 + ( totalInf + old\_LmdM ) * totalInf ) / Vtotal$  **Why not current Lambda here?** **This seems incorrect bug?**
9. Now calculate the wake skew parameter  $X_w$ :
  - a. IF  $TotalInf == 0$  THEN
    - i.  $xkai (\text{DynInflow})=0$
  - b. ELSE:
    - i.  $xkai = \text{TAN}( .5 * \text{ATAN}( \text{SQRT}( Vplane2 ) / totalInf ) )$
10. Get the current time step [L] matrices and invert them:
  - a. Call LMATRIX(  $xKai$ , 1) (*AeroSubs*) ; these are the L\_cos matrix
  - b. Call LMATRIX(  $xKai$ , 2) ; these are the L\_sin matrix
  - c. Call MATINV (  $xLcos$ ,  $xLsin$ ,  $maxInfl$ ,  $maxInfl0$ , 1) (*AeroSubs*) ;  $xLcos$  are now the  $L_{cos}^{-1}$  matrix, i.e.  $[\tilde{L}^c]^{-1}$
  - d. Call MATINV(  $xLcos$ ,  $xLsin$ ,  $maxInfl$ ,  $maxInfl0$ , 2) ;  $xLsin$  are now, the  $L_{sin}^{-1}$  matrix, i.e.  $[\tilde{L}^s]^{-1}$
11. Now calculate the  $\tau_n^{mc}$   $\tau_n^{ms}$ : start with the m=0 mode for cosines
  - a. DO\_LOOP mode = 1,  $maxInfl0 (\text{DynInflow})$ 
    - i.  $tauCos(mode) (\text{DynInflow})= - \Phi LqC(mode) (\text{DynInflow})/ Pzero * .5$
  - b. END\_LOOP; Note:  $\Phi LqC$   $\Phi LqS$  are calculated by GetPhiLq (*AeroSubs*) called by Inflow(*AeroSubs*) before this routine

- c. DO\_LOOP mode =  $\maxInfl0 + 1$ ,  $\maxInfl(\text{DynInflow})$ ; the other terms both cos and sin for m>0
- $\tau_{\text{Cos}}(\text{mode}) = -\Phi LqC(\text{mode}) / P_{\text{zero}}$
  - $\tau_{\text{Sin}}(\text{mode}) (\text{DynInflow}) = -\Phi LqS(\text{mode}) (\text{DynInflow}) / P_{\text{zero}}$
- d. END\_LOOP
12. Calculate the RHS of the state equations:  $-[L^c]^{-1}\{\alpha_l^k\} + \left\{\frac{\tau_n^{mc}}{2}\right\}$
- $RHS_{\text{cos}}(1) = 0.$
  - DO\_LOOP k = 1,  $\maxInfl$ ; this is for m=0 & n=1
    - $RHS_{\text{cos}}(1) = RHS_{\text{cos}}(1) + xL_{\text{cos}}(1,k) (\text{DynInflow}) * \text{old\_Alph}(k) (\text{DynInflow})$
  - END\_LOOP
  - $RHS_{\text{cos}}(1) = .5 * \tau_{\text{cos}}(1) - V_{\text{total}} * RHS_{\text{cos}}(1)$ ; note  $V_{\text{total}}$  for (m,n)=(0,1)
  - DO\_LOOP i = 2,  $\maxInfl0$ ; this is for m=0 & n>1
    - $RHS_{\text{cos}}(i) = 0.$
    - DO\_LOOP k = 1,  $\maxInfl$ 
      - $RHS_{\text{cos}}(i) = RHS_{\text{cos}}(i) + xL_{\text{cos}}(i,k) * \text{old\_Alph}(k)$
    - END\_LOOP
    - $RHS_{\text{cos}}(i) = .5 * \tau_{\text{cos}}(i) - V_{\text{param}} * RHS_{\text{cos}}(i)$ ; note  $V_{\text{param}}$  for (m,n)>(0,1)
  - END\_LOOP
  - DO\_LOOP i =  $\maxInfl0+1$ ,  $\maxInfl$ 
    - $RHS_{\text{cos}}(i) = 0.$
    - $RHS_{\text{sin}}(i) = 0.$
    - DO\_LOOP k = 1,  $\maxInfl$ 
      - $RHS_{\text{cos}}(i) = RHS_{\text{cos}}(i) + xL_{\text{cos}}(i,k) * \text{old\_Alph}(k)$
    - END\_LOOP
    - DO\_LOOP k =  $\maxInfl0 + 1$ ,  $\maxInfl$ ; now take care of  $-[L^s]^{-1}\{\beta_l^k\} + \left\{\frac{\tau_n^{ms}}{2}\right\}$ 
      - $RHS_{\text{sin}}(i) = RHS_{\text{sin}}(i) + xL_{\text{sin}}(i,k) (\text{DynInflow}) * \text{old\_Beta}(k) (\text{DynInflow})$
    - END\_LOOP
    - $RHS_{\text{cos}}(i) = .5 * \tau_{\text{cos}}(i) - V_{\text{param}} * RHS_{\text{cos}}(i)$
    - $RHS_{\text{sin}}(i) = .5 * \tau_{\text{sin}}(i) - V_{\text{param}} * RHS_{\text{sin}}(i)$
  - END\_LOOP
  - Now add the contribution from matrix [M] to RHS of GDW equations, calculated by *InflInit (AeroSubs)*
    - DO\_LOOP i =  $\maxInfl0$ ; for m=0
      - $d\text{Alph}_dt(i,1) (\text{DynInflow}) = xMinv(i) (\text{DynInflow}) * RHS_{\text{cos}}(i)$
    - END\_LOOP
    - DO\_LOOP i =  $\maxInfl0+1$ ,  $\maxInfl$ ; for m>0
      - $d\text{Alph}_dt(i,1) = xMinv(i) * RHS_{\text{cos}}(i)$
      - $d\text{Beta}_dt(i,1) (\text{DynInflow}) = xMinv(i) * RHS_{\text{sin}}(i)$
    - END\_LOOP

13. Now execute integration via Adams/Bashforth method of the GDW equations: Note the time-step here may be variable due to normalization if RPM changes from time-step to time-step: there may be problems integrating GDW
- Call *ABPRECOR*( *xAlpha*, *old\_Alph*, *dAlph\_dt*, *DT0*, *maxInfl*, 1 ) (*AeroSubs*)
  - Call *ABPRECOR*( *xBeta*, *old\_Beta*, *dBeta\_dt*, *DT0*, *maxInfl*, *maxInfl0+1* )
14. Calculate the new value for the induced velocity (normalized)  $\lambda_m$  normal to the rotor plane: the calculation makes use of the following:  $\lambda_m = \frac{2}{\sqrt{3}} \{1 \ 0 \ \dots \ 0\} [\tilde{L}^c]^{-1} \{\alpha_l^k\}$ , so only the first row of  $[\tilde{L}^c]^{-1}$  participates.
- Initialize: *xLambda\_M* = 0. (*DynInflow*)
  - DO\_LOOP *k* = 1, *maxInfl*; THIS IS POORLY WRITTEN IN THE CODE, POOR EFFICIENCY, MINOR BUG
    - xLambda\_M* = *xLambda\_M* + *xLcos(1,k)* \* *xAlpha(k)*
  - END\_LOOP
  - xLambda\_M* = 2. / sqrt(3.) \* *xLambda\_M*

## OUTPUT

It sets the following arrays and variables in *DynInflow*:

*DT0*: R(ReKi); non-dimensional time,  $DT^*\Omega$

*Pzero*: R(ReKi); normalization factor

*TipSpeed*: R(ReKi); tip-speed updated to the current time step

*totalInf*: R(ReKi); total wind velocity normal to rotor plane (induction included)

*Vtotal* : R(ReKi); total wind velocity magnitude at the rotor (induction included)

*Vparam* : R(ReKi); the mass-flow parameter

*xkai* : R(ReKi); the wake-skew parameter

*dAlph\_dt(1:maxInfl,1:4)* ; *dBeta\_dt(maxInfl0+1:maxInfl,1:4)* : R(ReKi) ; the time derivatives of the states.

*xLambda\_M*: R(ReKi); normalized, induced velocity normal to rotor plane

*xAlpha(maxInfl)* : R(ReKi); the alpha states at the current time step

*xBeta(maxInfl0+1:maxInfl)* : R(ReKi); the beta states at the current time step

It also sets the [L\_cos] and [L\_sin] matrices calling *LMATRIX* (*AeroSubs*) and inverte them calling them the same way: *xLcos* and *xLsin*.

## C34. LMATRIX(X,MATRIXMODE)

Subroutine that calculates the L\_cos and L\_sin matrices, using *Gamma* and  $xKai=X_w$ . These are the  $[L^c]$  and  $[L^s]$ . See Theory Sections.

Called by: *InflInit*, *InflDist* (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

- *DynInflow* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
X	R(ReKi)	IN	This is the wakeskew parameter
<i>matrixMode</i>	I(4)	IN	Flag to indicate whether to calculate L_cos (1) or L_sin (2)

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
XM	R(ReKi)			$X^{**m}$ (see Theory)
IROW	I			Counter for rows
JCOL	I			Counter for columns

### STEPS

1. IF  $X < -1 .OR. X > 1$  THEN: NOT SURE WHY THIS CONDITION, it should be  $0 < (X = \tan(\chi/2))$ 
  - a. Warn user and EXIT
2. Based on *matrixMode*:
  - a. =1: calculate L\_cos
    - i. DO\_LOOP  $JCOL = 1, maxInfl$  (*DynInflow*);
      1.  $XM = X^{**MRvector}(JCOL)$  (*DynInflow*); note auxiliary matriced defined in *InflInit* (*AeroSubs*)
      2. DO\_LOOP  $IROW = 1, maxInfl0$  (*DynInflow*); this is for  $k=0$  states
        - a.  $xLcos(IROW, JCOL) = GAMMA(IROW, JCOL)$  (*DynInflow*) \*  $XM$ ; note *GAMMA* was calculated by *InflInit* (*AeroSubs*)
      3. END\_LOOP
    - ii. END\_LOOP
    - iii. DO\_LOOP  $IROW = maxInfl0 + 1, maxInfl$ ; Here take care of  $k > 0$  states
      1. DO\_LOOP  $JCOL = 1, maxInfl$ 
        - iv.  $xLcos(IROW, JCOL) = GAMMA(IROW, JCOL) * [X^{**MminusR}(IROW, JCOL) + X^{**MplusR}(IROW, JCOL) * (-1)^{**MminR}(IROW, JCOL)]$
      - v. END\_LOOP

- b. =2: calculate L\_sin
  - i. DO\_LOOP IROW = maxInfl0+1, maxInfl
    - 1. DO\_LOOP JCOL = maxInfl0+1, maxInfl
      - a.  $xLsin(IROW, JCOL) = GAMMA(IROW, JCOL) * [X^{MminusR(IROW, JCOL)} - X^{MplusR(IROW, JCOL)} * (-1)^{MminR(IROW, JCOL)}]$
    - 2. END\_LOOP
  - ii. END\_LOOP
- c. All other cases: warn user of wrong input *matrixMode* and EXIT

## OUTPUT

It outputs  $xLcos(maxinfl, maxinfl)$ ,  $xLsin(maxInfl0+1:maxInfl, maxInfl0+1:maxInfl)$ : R(ReKi); i.e.  $[\tilde{L}_{ln}^{km}]^c$  and  $[\tilde{L}_{ln}^{km}]^s$ .

## C35. GETPHILQ()

Subroutine that calculates the integral of blade elemental forces times the radial shape function  $\varphi_n^m$ , along the span and for all of the blades. It is needed in the calculation of  $\tau_n^{mc}$  and  $\tau_n^{ms}$ . See Theory.  $\Phi LqC$  is the integral for cosine terms.  $\Phi LqS$  is the analogous for sine terms.

Called by: *Inflow*, (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

- *Blade* [see AeroMods.f90]
- *DynInflow* [see AeroMods.f90]
- *Element* [see AeroMods.f90]

### INPUT

Additional Input via invoked module variables (e.g., *Nb (Blade)* *MaxInfl (DynInflow)*).

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>iblad</i>	I(4)			Blade counter
<i>ielem</i>	I(4)			Element counter
<i>mode</i>	I(4)			Counter for modes

### STEPS

1. Initialize the following two variables belonging to *DynInflow*:
  - a.  $\Phi LqC = 0$ .
  - b.  $\Phi LqS = 0$ .
2. DO\_LOOP *mode*=1, *MAXINFL (DynInflow)*: for all the states (modes) available (6 currently), perform the integrals along the blade spans and sum contributions from all blades.
  - a. DO\_LOOP *iblad*=1, *Nb (Blade)*; cycle on blades
    - i. DO\_LOOP *ielem*=1, *NELM (Element)*; cycle on blade elements
      1.  $\Phi LqC(\text{mode}) = \Phi LqC(\text{mode}) + RMC\_SAVE(iblad, ielem, mode)$  (*DynInflow*): for cosine terms. Note *RMC\_SAVE* (BE force times radial shape function) was calculated by *GET RM (AeroSubs)* which was called by *AD\_CalculateLoads (AeroDyn)* at time step 0 as well.
      2. IF (*mode* >= *maxInfl0*+1) THEN: now we passed the first m=0 modes, and we can think about sine terms as well
        - a.  $\Phi LqS(\text{mode}) = \Phi LqS(\text{mode}) + RMS\_SAVE(iblad, ielem, mode)$
        - ii. END\_LOOP
    - b. END\_LOOP
  3. END\_LOOP

### OUTPUT

*PhiLqC(MaxInfl), PhiLqS(MaxInfl): R(Reki) (DynInflow):* these are the integrals of the mode-moments of the blade element forces, needed to calculate  $\tau_n^{mc}$  and  $\tau_n^{ms}$ .

## C36. WINDAZIMUTHZERO (PSI,WINDPSI)

Subroutine added by JRS to define the zero azimuth datum in a wind based co-ordinate system, for use in the dynamic inflow routines. Called by GetRM, VindInf (*AeroSubs*).

### THEORY

It subtracts from the blade azimuth the angle formed by the in-plane component of the wind velocity with the shaft reference frame negative z-axis (-Z<sub>FFs</sub>). It uses the two vectors: *VrotorZ* and *VrotorY*. *VrotorZ* is positive vertically upwards and negative vertically downwards, *VrotorY* is positive to the left and negative to the right, both when looking towards the rotor from upwind. They are wind velocity components along the shaft reference frame, calculated by DiskVel (*AeroSubs*) called by AD\_CalculateLoads (*Aerodyn*). Zero-azimuth is defined when vertically down, positive clockwise.

It basically attempts at restoring the helicopter reference frame FF<sub>HR</sub>.

### EXTERNAL (INVOKED) MODULES

- *Wind* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
<i>psi</i>	R(ReKi)	IN	Azimuth in
<i>WindPsi</i>	R(ReKi)	OUT	Azimuth out

Additional variables via invoked module (e.g. *VRotory*)

### LOCAL VARIABLES

N/A

### STEPS

1.  $\text{WindPsi} = \text{psi} - \text{ATAN2}(\text{VrotorY}(\text{Wind}), -\text{VrotorZ}(\text{Wind}))$

### OUTPUT

*WindPsi*: real(ReKi). It calculates azimuth in a wind reference frame with zero at bottom and positive clockwise.

## C37. XPHI(RZERO,MODE)

Function that calculates the values of the radial shape function at the local blade-element non-dimensional radial distance ( $Rzero$ ): it considers only states that would yield a maximum power of 3 for  $Rzero$ . This is equivalent to 10 total states. Called by *GetRM*, *VindInf*(*AeroSubs*).

### THEORY

Carrying out the math one obtains for the various functions with maximum power of 3:

$$\begin{aligned}\varphi_1^0(\hat{\mathbf{r}}_e(\mathbf{v})) &= \sqrt{3.} && ; \text{for } (m,n)=(0,1) \\ \varphi_3^0(\hat{\mathbf{r}}_e(\mathbf{v})) &= 2\sqrt{7}(1.5 - 3.75 Rzero^{**2}) / 3. && ; \text{for } (m,n)=(0,3) \\ \varphi_2^1(\hat{\mathbf{r}}_e(\mathbf{v})) &= \sqrt{15./2.} * Rzero && ; \text{for } (m,n)=(1,2) \\ \varphi_4^1(\hat{\mathbf{r}}_e(\mathbf{v})) &= 4*(15/4 * Rzero - 105/16 * Rzero^{**3})/\sqrt{5} && ; \text{for } (m,n)=(1,4) \\ \varphi_3^2(\hat{\mathbf{r}}_e(\mathbf{v})) &= \sqrt{105./2.} / 2. * Rzero^{**2} && ; \text{for } (m,n)=(2,3) \\ \varphi_4^3(\hat{\mathbf{r}}_e(\mathbf{v})) &= \sqrt{35.} * 3. / 4. * Rzero^{**3} && ; \text{for } (m,n)=(3,4)\end{aligned}$$

### EXTERNAL (INVOKED) MODULES

N/A

### INPUT

Name	Type	INTENT	Description
<i>Rzero</i>	R(ReKi)	IN	Non-dimensional radial coordinate
<i>mode</i>	I(4)	IN	Counter for the

### LOCAL VARIABLES

N/A

### STEPS

1. IF  $Rzero < 0$ . OR.  $Rzero > 1$  THEN
  - a. Warn user and EXIT program
2. Based on mode:
  - a. =1:  $xphi = 1.732051$  ;for  $(m,n)=(0,1)$
  - b. =2:  $xphi = 2.645751 - 6.6143783 * Rzero^{**2}$  ;for  $(m,n)=(0,3)$
  - c. =3:  $xphi = 2.738613 * Rzero$  ;for  $(m,n)=(1,2)$
  - d. =4:  $xphi = (6.708204 - 11.73936 * Rzero^{**2}) * Rzero$  ;for  $(m,n)=(1,4)$
  - e. =5:  $xphi = 3.622844 * Rzero^{**2}$  ;for  $(m,n)=(2,3)$
  - f. =6:  $xphi = 4.437060 * Rzero^{**3}$  ;for  $(m,n)=(3,4)$
  - g. All other cases: warn user and EXIT program.
  - h. Note there is a function PHIS (*AeroSubs*) that calculates the actual  $\varphi_l^k$  as a function of generic indices  $k$  and  $l$ .

### OUTPUT

$Xphi$ : R(ReKi). These are the values of the radial shape function for the requested  $(m,n)$ -combo at the local blade element radial distance.

## C38. MATINV (A0, A1, N, N0, INVMODE)

Subroutine that inverts the [L\_cos] and [L\_sin] matrices. Also see function [GAUSSI](#) (*AeroSubs*).

Called by: [InflInit](#), [Infdist](#) (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

N/A

#### INPUT

Name	Type	INTENT	Description
<i>invMode</i>	I(4)	IN	Switch for [L_cos] (1) or [L_sin] (2) matrix
<i>N</i>	I(4)	IN	Number of rows/columns for [L_cos]
<i>N0</i>	I(4)	IN	Start index for [L_Sin] ( <i>MaxInfl0 DynInlfow AeroMods.[f90]</i> )
<i>A0(N,N)</i>	R(ReKi)	INOUT	IN: [L_cos]; OUT: Square matrix inverse of [L_cos]
<i>A1(N0+1:N,N0+1:N)</i>	R(ReKi)	INOUT	IN: [L_sin]; OUT Square matrix inverse of [L_sin]

#### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>DUMMY(N-N0,N-N0)</i>	R(ReKi)			Temporary matrix
<i>I</i>	I(4)			Loop counter for rows
<i>J</i>	I(4)			Loop counter for columns

#### STEPS

1. Based on *invMode*:
  - a. =1: Call [GAUSSI](#)(*A0,N*) (*AeroSubs*) ;Invert [L\_cos] by Gauss-Jordan method
  - b. =2: Use a *DUMMY* matrix to recast
    - i. DO\_LOOP *I=1, N-N0*
      1. DO\_LOOP *J=1, N-N0*
        - a. *DUMMY(I,J) = A1(I+N0,J+N0)*
      2. END\_LOOP
    - ii. END\_LOOP
    - iii. Now invert the *DUMMY* matrix : Call [GAUSSI](#)(*DUMMY,N-N0*) (*AeroSubs*)
    - iv. Now revert back to *A1*:
      1. DO\_LOOP *I=1, N-N0*
        - a. DO\_LOOP *J=1, N-N0*
          - i. *A1(I+N0,J+N0) = DUMMY(I,J)*
        - b. END\_LOOP

2. END\_LOOP; Note sure if this is needed at all, in fact one could pass  
A2=A1(N0:N,N0:N) to GAUSSJ, Minor Bug that slows down the prg
- c. All other cases: warn user and EXIT program.

## OUTPUT

It sets  $A0(N,N)$ ,  $A1(N0+1:N,N0+1:N)$  : real(ReKi); i.e.  $[L_{\cos}]^{-1}$   $[L_{\sin}]^{-1}$ .

## C39. GAUSSJ (A, N)

Subroutine that inverts matrix A(N,N) via Guass\_Jordan method.

Called by: MATINV (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

N/A

#### INPUT

Name	Type	INTENT	Description
<i>N</i>	I(4)	IN	Number of rows/columns matrix
<i>A(N,N)</i>	R(ReKi)	INOUT	IN: matrix; OUT: inverse of matrix

#### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>NMAX</i>	I(4)	6	PARAMETER	Max size of allowed square matrix(Nmax x Nmax)
<i>Big</i>	R(ReKi)			Aux variable
<i>dum</i>	R(ReKi)			Aux variable
<i>pivinv</i>	R(ReKi)			Aux variable
<i>i</i>	I(4)			Loop counter
<i>icol</i>	I(4)			Index of column
<i>indx(NMAX)</i>	I(4)			Aux array
<i>indx(NMAX)</i>	I(4)			Aux array
<i>ipiv(NMAX)</i>	I(4)			Aux array
<i>irow</i>	I(4)			Index of row
<i>j</i>	I(4)			Loop counter
<i>k</i>	I(4)			Loop counter
<i>l</i>	I(4)			Loop counter
<i>ll</i>	I(4)			Loop counter

#### STEPS

1. Initialize *ipiv*: *ipiv*(1:N)=0 :why the Do\_Loop? Minor Bug slowing down the prg
  - a. DO\_LOOP *j*=1,*N*
    - i. *ipiv(j)*=0

- b. END\_LOOP
- 2. DO\_LOOP  $i=1, N$ 
  - a. Set  $big=0$
  - b. DO\_LOOP  $J=1, N:$  rows
    - i. IF  $ipiv(j) < 1$  THEN:
      - 1. DO\_LOOP  $k=1, N:$  columns
        - a. IF  $ipiv(k) == 0$  THEN
          - i. IF  $|A(j,k)| >= big$  THEN
            - 1.  $big=|A(j,k)|$
            - 2.  $irow=j$
            - 3.  $icol=k$
        - b. ELSEIF  $ipiv(k) > 1$  THEN
          - i. Warn User of Singular Matrix and EXIT
      - 2. END\_LOOP
    - c. END\_LOOP
    - d.  $ipiv(icol)=ipiv(icol)+1$
    - e. IF  $(irow < col)$  THEN
      - i. DO\_LOOP  $l=1, N:$  scan columns
        - 1.  $dum=A(irow,l)$
        - 2.  $A(irow,l)=A(icol,l)$
        - 3.  $A(icol,l)=dum$
      - ii. END\_LOOP
    - f. Set  $idxr(i)=irow$
    - g. Set  $idxc(i)=icol$
    - h. IF  $A(icol,icol) == 0$ . THEN: Warn User of Singular Matrix and EXIT
    - i. Set  $pivinv=1./A(icol,icol)$
    - j.  $A(icol,icol)=1.$
    - k. DO\_LOOP  $l=1,N$ 
      - l.  $A(icol,l)=A(icol,l)*pivinv$
    - m. END\_LOOP
    - n. DO\_LOOP  $ll=1,N$ 
      - i. IF  $ll < col$  THEN
        - 1.  $dum=A(ll,icol)$
        - 2.  $A(ll,icol)=0.$
        - 3. DO\_LOOP  $l=1,N$ 
          - a.  $A(ll,l)= A(ll,l)-A(icol,l)*dum$
      - 4. END\_LOOP
    - o. END\_LOOP
  - 3. END\_LOOP
  - 4. DO\_LOOP  $l=N,1,-1$

- a. IF  $idxr(l) < > idxc(l)$  THEN
  - i. DO\_LOOP  $k=1, N$ : columns
    1.  $dum=A(k,idxr(l))$
    2.  $A(k,idxr(l))=A(k,idxc(l))$
    3.  $A(k,idxc(l))=dum$
  - ii. END\_LOOP
5. END\_LOOP

**OUTPUT**

It outputs  $A(N,N)$ : real(ReKi); i.e. the inverse of input  $A(N,N)$ .

## C40. FGAMMA (R,J,M,N)

Function that calculates the function  $\Gamma_{jn}^{rm}$ , with generic indices which is used in the calculation of the [L] matrices. See Theory Sections. Note  $r, m$  are indices referring to harmonics in azimuth, and  $j, n$  to radial expansions.

Called by: *Infinit (AeroSubs)*.

### EXTERNAL (INVOKED) MODULES

N/A

### INPUT

Name	Type	INTENT	Description
$J$	I(4)	IN	Index of radial expansion mode
$M$	I(4)	IN	Index of harmonic
$N$	I(4)	IN	Index of radial expansion mode
$R$	I(4)	IN	Index of harmonic

### LOCAL VARIABLES

N/A

### STEPS

1. IF MOD( $R+M,2$ ) == 0 THEN : r+m even
  - a.  $FGAMMA = (-1)^{**((N+J-2*R)*.5)} * 2.*SQRT[ REAL( (2*N+1) * (2*J+1) ) ] &$   
 $/ [SQRT( \underline{HFUNC}(M,N) * \underline{HFUNC}(R,J) (AeroSubs) ) * REAL( (J+N) * (J+N+2) * ((J-N)**2-1) ] ]$
2. ELSEIF  $| J-N | == 1$  THEN: (also r+m odd)
  - a.  $FGAMMA = 3.14159265 * SIGN(1., REAL(R-M) ) * .5 /$   
 $[ SQRT( \underline{HFUNC}(M,N) * \underline{HFUNC}(R,J) ) * SQRT( REAL( (2*N+1) * (2*J+1) ) ) ]$
3. ELSE
  - a.  $FGAMMA = 0$

### OUTPUT

It outputs  $FGAMMA$ : R(ReKi); i.e. the value of the function  $\Gamma_{jn}^{rm}$ .

## C41. HFUNC (M,N)

Function that calculates the value of the function  $H_n^m$ , with generic indices  $m,n$ ; it is used in the calculation of the  $\Gamma_{ln}^{km}$  function as well as [M] matrices. See Theory Sections. Note  $m$  index refers to harmonics in azimuth, and  $n$  to radial expansion modes.

Called by: *Infinit*, *PHIS*, *FGAMMA*, (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

N/A

### INPUT

Name	Type	INTENT	Description
$M$	I(4)	IN	Index of harmonic
$N$	I(4)	IN	Index of radial expansion mode

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
$NMM$	I(4)			$N-M$
$NPM$	I(4)			$N+M$

### STEPS

3. IF  $N \leq M$  THEN:
  - a. Warn user and EXIT
4. Set  $NPM = N+M$
5. Set  $NMM = N-M$
6. Set Output:  $HFUNC = \text{REAL}(\underline{\text{IDUBFACT}}(NPM-1)) / \text{REAL}(\underline{\text{IDUBFACT}}(NPM))$   
 $* \text{REAL}(\underline{\text{IDUBFACT}}(NMM-1)) / \text{REAL}(\underline{\text{IDUBFACT}}(NMM))$

### OUTPUT

It outputs  $HFUNC$ : R(ReKi); i.e. the value of the function  $H_n^m$ .

## C42. IDUBFACT (I)

Function that calculates the double factorial (see Theory Sections).

Called by: PHIS, HFUNC, (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

N/A

### INPUT

Name	Type	INTENT	Description
<i>I</i>	I(4)	IN	Intger number

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>K</i>	I(4)			Counter

### STEPS

1. IF  $I \geq 1$  THEN:
  - a. Initialize:  $IDUBFACT = 1$
  - b. DO  $K = I, 1, -2$ 
    - i.  $IDUBFACT = IDUBFACT * K$
    - c. END\_LOOP
2. ELSEIF  $I == 0 .OR. I == -1$  THEN
  - a.  $IDUBFACT = 1$
3. ELSEIF  $I == -3$  THEN
  - a.  $IDUBFACT = -1$
4. ELSE:
  - a. Warn user of undefined double factorial for numbers <3 and EXIT

### OUTPUT

It outputs  $IDUBFACT$ : I(4); i.e. the value of the double factorial for an integer.

## C43. PHIS (RZERO, R, J)

Function that calculates the  $\varphi_j^r(\hat{r}(v))$  function for generic indices  $r$  (harmonic mode) and  $j$  (radial shape modes) (see Theory Sections).

Called by: external process, or if AeroDyn changes, by other internal routines in case more than current 10 states are used.

### EXTERNAL (INVOKED) MODULES

N/A

#### INPUT

Name	Type	INTENT	Description
$Rzero$	R(ReKi)	IN	Non-dimensional radial distance
$j$	I(4)		Index for radial shape mode
$r$	I(4)		Index for azimuth harmonic mode

#### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
$q$	I(4)			Counter

#### STEPS

1. IF  $Rzero < 0 .OR. > 1$  THEN (Could be better written, minor bug)
  - a. Warn user of wrong input and EXIT
2. Initialize  $PHIS=0$ .
3. DO\_LOOP  $q=r, j-1, 2$ 
  - a.  $PHIS = PHIS + Rzero ** q * (-1)**((q-r)/2) * [ \text{REAL}(\underline{\text{IDUBFACT}}(j+q\_) \& / \text{REAL}(\underline{\text{IDUBFACT}}(q-r\_) * \underline{\text{IDUBFACT}}(q+r\_) * \underline{\text{IDUBFACT}}(j-q-1\_) ] ) (AeroSubs)$
4. END\_LOOP
5. Final Output:  $PHIS = PHIS * \text{SQRT}[ \text{REAL}(2*j+1) * \underline{\text{HFUNC}}(r,j\_) ] (AeroSubs)$

#### OUTPUT

It outputs  $PHIS$ : R(ReKi); i.e. the value of the  $\varphi_j^r(\hat{r}(v))$  at the local blade element radial distance (normalized by blade  $R$ ).

## C44. VINDINF (IRADIUS, IBLADE, RLOCAL, VNW, VNB, VT, PSI)

Subroutine that calculates the axial induction factor for each BE position using the calculated GDW parameters.

Called by *ElemFrc* (*AeroSubs*) for each element at a new time step.

### EXTERNAL (INVOKED) MODULES

- *DynInflow* [see AeroMods.f90]
- *Blade* [see AeroMods.f90]
- *Element* [see AeroMods.f90]
- *Switch* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
<i>iradius</i>	I	IN	BE index counter
<i>iblade</i>	I	IN	Blade index counter
<i>Rlocal</i>	R(ReKi)	IN	Local BE radial distance from rotational axis
<i>VNW</i>	R(ReKi)	IN	wind velocity component normal to the rotor plane
<i>VNB</i>	R(ReKi)	IN	Body motion velocity component normal to the plane of the rotor
<i>VT</i>	R(ReKi)	INOUT	Tangential velocity due to rigid and elastic motions+wind
<i>psi</i>	R(ReKi)	IN	Azimuth angle

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>A2P</i>	R(ReKi)			
<i>Rzero</i>	R(ReKi)			<i>Rlocal/R(Blade)</i>
<i>SWRLARG</i>	R(ReKi)			
<i>Windpsi</i>	R(ReKi)			Modified azimuth to account for FFHR system of reference
<i>mode</i>	I(4)			State counter

### STEPS

1. Set non-dimensional radial distance:  $Rzero = rlocal / r$
2. Adjust azimuth to account for FFHR reference frame:
  - a. Call *WindAzimuthZero*(*psi, WindPsi*) (*AeroSubs*)

3. Calculate the induction factor using the GDW parameters: from definition of induced velocity component normal to rotor as expansion of harmonics and radial shape functions (see Theory)
  - a. Initialize:  $A(iRadius,iBlade) = 0.$  (*Element*)
  - b. Sum all harmonics and shape functions to get the normalized component of the induced velocity normal to rotor:
    - i. DO\_LOOP mode = 1,  $maxInfl0$  (*DynInflow*) ; for m=0
      1.  $A(iRadius,iBlade)=A(iRadius,iBlade)+xphi(Rzero,mode)(AeroSubs)$  \*  $xAlpha(mode)$  (*DynInflow*)
    - ii. END\_LOOP
    - iii. DO\_LOOP mode =  $maxInfl0+1, maxInfl$  (*DynInflow*) ; for m>0
      1.  $A(iRadius,iBlade) = A(iRadius,iBlade) + xphi(Rzero,mode) * [ xAlpha(mode) * \cos(\text{REAL}(MRvector(mode))(\text{DynInflow}) * Windpsi) + xBeta(mode)(\text{DynInflow}) * \sin(\text{REAL}(MRvector(MODE)) * Windpsi) ]$
    - iv. END\_LOOP
  - c. Now,  $a$  can be calculated from the induced velocity, de-normalizing, and dividing by the incoming wind velocity normal to the rotor plane (VNW): This should account for body motion too to be consistent with the tangential portion of relative air-velocity
    - i.  $A(iRadius,iBlade) = - A(iRadius,iBlade) * TipSpeed / VNW$ ; Note the sign here is a bit confusing, suggest changing sign conventions per Peters (2011)
4. Now calculate the tangential induction factor  $a'$ :
  - a. IF SWIRL (*Switch*) THEN : rotation wake to be included in calculations: Note Goldstein's hypothesis used, with the added oddity of being inconsistent between induction in the axial and tangential direction for the body motions.
    - i.  $SWRLARG=1+4*A(iradius,iblade)*VNW * [ (1.0 - A(iradius,iblade))*VNW + VNB]/VT^2$
    - ii. IF  $SWRLARG > 0$ . THEN: do not bother if the argument is  $<=0$ 
      1.  $A2P$  (*Element*)= $0.5 * (-1.0 + \sqrt(SWRLARG))$
      2.  $VT = VT * ( 1.0 + A2P)$ ; Not sure why this gets updated here, but it always does

## OUTPUT

It sets  $A(iRadius,iBlade)$   $A2P(iRadius,iBlade)$ : R(ReKi); i.e. axial and induction factors starting from GDW.

It sets  $VT$ : R(ReKi); the tangential velocity component contributed by wind, rigid and elastic motions.

## C45. DYNDEBUG (RHScos, RHSSIN)

Subroutine that writes GDW current states to a file for debugging purposes.

Called by: external process. It could be called from *InfDist (AeroSubs)*.

### EXTERNAL (INVOKED) MODULES

- *AeroTime* [see AeroMods.f90]
- *DynInflow* [see AeroMods.f90]

### INPUT

Name	Type	INTENT	Description
<i>RHScos (maxInfl (DynInflow))</i>	R(ReKi)	IN	Never used here
<i>RHSSin (maxInfl0+1:maxInfl (DynInflow))</i>	R(ReKi)	IN	Never used here

Additional variables via invoked modules (e.g., *TIME (AeroTime)*).

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>i</i>	I(4)			Counter
<i>NumOut</i>	I(4)			<i>maxInfl + (maxInfl-maxInfl0) + 1 (DynInflow)</i>
<i>UnDyn</i>	I(4)	80	Initialized	File unit for output
<i>OnePass</i>	L	.TRUE.		If first pass at outputting the debug info
<i>Frmt</i>	C(50)			Format string

### STEPS

1. Initialize: *NumOut* = *maxInfl + (maxInfl-maxInfl0) + 1 (DynInflow)*
2. IF *OnePass* THEN
  - a. Open File for output info: Call *OpenFOutFile (UnDyn, 'DynDebug.plt')* (*NWTC\_IO*)
  - b. Set the format string accounting for *NumOut* for the header
  - c. Write to *UnDyn* the header made up of Time, 'dAlpha\_dt', 'dBeta\_dt', 'TotalInf', repeated the right number of times, i.e. *dAlpha\_dt* will have *maxInfl* numbers and *dBeta\_dt* *maxInfl - maxInfl0*
  - d. Set *OnePass* = .FALSE.
3. Set a new format string for the actual values of the variables
4. IF (*TIME (AeroTime)* > 0.0D0) THEN
  - a. Write to *UnDyn* the values of *TIME*, *dAlpha\_dt (DynInflow)*, *dBeta\_dt (DynInflow)*, *TotalInf (DynInflow)*, repeated the right number of times, i.e. *dAlpha\_dt* will have *maxInfl* numbers and *dBeta\_dt (maxInfl - maxInfl0)*

### OUTPUT

It writes the values of the current *TIME(AeroTime)*, *dAlpha\_dt (DynInflow)*, *dBeta\_dt (DynInflow)*, *TotalInf (DynInflow)* to a file for debugging purposes.

## C46. INFUPDT ()

Subroutine that updates the states of GDW, i.e.  $dAlpha_dt$  and  $dBeta_dt$ .

Called by: *Inflow* (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

- *DynInflow* [see AeroMods.f90]

### INPUT

Variables from invoked modules (e.g., *old\_Alph*, *old\_Beta*, *dAlpha\_Dt* (*DynInflow*)).

### LOCAL VARIABLES

Name	Type	Value	PARAMETER/ Initialized	Description
<i>i</i>	I(4)			Counter

### STEPS

1. Update the old values with current values of the following variables (*DynInflow*)
  - a.  $oldKai = xKai$
  - b.  $old\_LmdM = xLambda\_M$
2. Update the states, first those that work with cosines, where we have the  $m=0$  states as well (in the basic case  $n=1$  and  $n=3$ ); also update the previous 3 values of the states at the previous time steps for Adams-Bashforth method
  - a. DO\_LOOP  $i = 1, maxInfl$  (*DynInflow*)
    - i.  $old\_Alph(i) = xAlpha(i)$
    - ii.  $dAlpha\_dt(i,4) = dAlpha\_dt(i,3)$
    - iii.  $dAlpha\_dt(i,3) = dAlpha\_dt(i,2)$
    - iv.  $dAlpha\_dt(i,2) = dAlpha\_dt(i,1)$
  - b. END\_LOOP
3. Now Update the states that work with sines, where we do not have the  $m=0$  states; also update the previous 3 values of the states at the previous time steps for Adams-Bashforth method
  - a. DO\_LOOP  $i = maxInfl+1, maxInfl$  (*DynInflow*)
    - i.  $old\_Beta(i) = xBeta(i)$
    - ii.  $dBeta\_dt(i,4) = dBeta\_dt(i,3)$
    - iii.  $dBeta\_dt(i,3) = dBeta\_dt(i,2)$
    - iv.  $dBeta\_dt(i,2) = dBeta\_dt(i,1)$
  - b. END\_LOOP

### OUTPUT

It sets the new values for the following *DynInflow* variables:

*old\_Alph(1:maxInfl)*, *dAlpha\_dt(1:maxInfl,2:4)* , *old\_Beta(maxInfl0+1:maxInfl)*, *dBeta\_dt(maxInfl0+1:maxInfl,2:4)*, *oldKai*, *old\_LmdM* : R(ReKi).

## APPENDIX D. AEROMODS MODULE

### GENERAL ORGANIZATION

AeroMods.f90 contains several modules whose only purposes are to define variables and constants (parameters).

A new type of wind field has been added to AeroDyn in v12.57. When *CTWindFlag* is .TRUE., coherent turbulence is added to a background wind field. Thus, one other XXWindFlag should also be .TRUE. when *CTWindFlag* is .TRUE. (Currently, *FFWindFlag* is the only type that can be set when *CTWindFlag* is set, but that may change in the future.)

### EXTERNAL (INVOKED) MODULES

N/A

### INTERNAL (LOCALLY DECLARED) MODULES

- *AD\_IOParams*: It contains 4 initialized integer(4) parameters (declared as in/out variables however), and 1 initialized logical parameter:

Name	Type	Value	Parameter/ Initialized	Description
<i>UnADin</i>	I(4)	90	PARAMETER	Ipt file unit
				Opt file unit, it is a sort of echo of the input file
<i>UnADopt</i>	I(4)	92	PARAMETER	with options selected, file names, and info on rotor radius, hub radius and number of blades
<i>UnAirfl</i>	I(4)	93	PARAMETER	Airfoil data file unit
<i>UnWind</i>	I(4)	91	PARAMETER	HH or FF wind file unit
<i>WrOptFile</i>	L	.TRUE.	Initialized	Write .opt file, Y vs N switch

- *AeroTime*: It contains invocation to Module *Precision* [see DoublePrec.f90] and 4 real variables:

Name	Type	Value	Parameter/ Initialized	Description
<i>OLDTIME</i>	R(ReKi)			Previous time Aerodyn's loads were calculated
<i>TIME</i>	R(DbKi)			Current time simulations time
<i>DT</i>	R(ReKi)			Actual difference between Time and OldTime when loads are calculated
<i>DTAERO</i>	R(ReKi)			Desired time interval for aerodynamics calculations

- **Switch:** It contains 16 logical variables to control program options:

Name	Type	Value	Parameter/ Initialized	Description
DSTALL	L			.TRUE.=Beddoes; .FALSE.=Steady
DYNINFL	L			.TRUE.=DYNIN; .FALSE.= Equil (BEM)
DYNINIT	L			It is set to .TRUE. if in need of initializing <b><i>InflInit</i></b> <b>(AeroSubs)</b>
ELEMPRN	L			If user selected to print element information
EquilDA	L			If .TRUE. THEN drag term included in the axial momentum equation
EquilDT	L			If .TRUE. THEN drag term included in the tangential (torque) momentum equation
GTECH	L			Georgia Tech Correction for tip-loss
HLOSS	L			Hub Loss: .TRUE.=PRAND vs .FALSE.=NONE
MultiTab	L			Whether we have multiple tables (e.g. Reynolds numbers) in airfoil files
PMOMENT	L			Pitching Moment; .TRUE.= USE_CM; .FALSE.=NO_CM
Reynolds	L			Whether we have multiple Reynolds numbers in airfoil files
SIUNIT	L			.TRUE.=SI; .FALSE.=English
SKEW	L			It is set if the in-rotorplane wind speed is >0.001 m/s
SWIRL	L			.TRUE.= tangential induction factor to be calculated
TLOSS	L			TIP Loss: .TRUE.=PRAND vs. .FALSE.=NONE
WAKE	L			.TRUE.= axial induction factor to be calculated

- **Blade:** It contains blade information: invocation to Module **Precision** [see DoublePrec.f90 ], 1 real variable, 2 real dynamic arrays, 1 integer variable

Name	Type	Value	Parameter/ Initialized	Description
C (:)	R(ReKi)			Array( <i>NElm (Element)</i> ) Chord of each blade element from input file

<i>DR</i> (:)	R(ReKi)	Array( <i>NElm (Element)</i> ) Spanwise length of blade element centered at Relm(:) from input file
<i>R</i>	R(ReKi)	Rotor radius
<i>NB</i>	I(4)	Number of Blades

- *Element*: It contains blade element information: invocation to Module *Precision* [see DoublePrec.f90 ], 1 real variable, 6 real dynamic arrays, 1 integer variable

Name	Type	Value	Parameter/ Initialized	Description
<i>A</i> (:,:)	R(ReKi)		array( <i>NElm (Element)</i> , <i>NB (Blade)</i> ): Axial Induction factor a, per blade and blade element (i,j)	
<i>AP</i> (:,:)	R(ReKi)		array( <i>NElm</i> , <i>NB</i> ): Tangential Induction factor a', per blade and blade element (i,j)	
<i>HLCNST</i> (:)	R(ReKi)		array( <i>NElm</i> ): Hub-loss constant at each blade element	
<i>RELM</i> (:)	R(ReKi)		array( <i>NElm</i> ): Radius location of each blade element, measured from blade root	
<i>TLCNST</i> (:)	R(ReKi)		array( <i>NElm</i> ): Tip-loss constant at each blade element	
<i>TWIST</i> (:)	R(ReKi)		array( <i>NElm</i> ): Twist angle of each blade element, from input file	
<i>PITNOW</i>	R(ReKi)			Pitch angle of the blade element
<i>NELM</i>	I(4)			Number of elements (per blade), from input file

- *ELEMInflow*: It contains blade element information associated with the inflow: invocation to Module *Precision* [see DoublePrec.f90 ], 2 real dynamic arrays:

Name	Type	Value	Parameter/ Initialized	Description
<i>ALPHA</i> (:,:)	R(ReKi)			Angle of attack of current blade element and time step
<i>W2</i> (:,:)	R(ReKi)			Relative Wind Speed Squared for current element and time

- **Rotor:** It contains rotor configuration information, , invocation to Module *Precision* [see DoublePrec.f90 ] and 10 real(ReKi) variables:

Name	Type	Value	Parameter/ Initialized	Description
AVGINFL	R(ReKi)			Average induced velocity at previous step
CTILT	R(ReKi)			Cosine of Tilt Angle
CYaw	R(ReKi)			Cosine of Yaw Angle
HH	R(ReKi)			HubHeight
REVS	R(ReKi)			Rotor rotational speed (i.e. RPM in rad/sec)
STILT	R(ReKi)			Sine of Tilt Angle
SYaw	R(ReKi)			Sine of Yaw Angle
TILT	R(ReKi)			Tilt Angle
YawAng	R(ReKi)			Yaw Angle
YAWVEL	R(ReKi)			Velocity of Hub along hub y-axis due to yaw-components of rotational velocities of turbine parts

- **InducedVel:** It contains induced velocity information, invocation to Module *Precision* [see DoublePrec.f90 ] and 3 real variables:

Name	Type	Value	Parameter/ Initialized	Description
ATOLER	R(ReKi)			Convergence tolerance for axial Induction factor a
EqAIDmult	R(ReKi)			Drag term multiplier in the axial-induction equation (either 0 or 1 depending on whether drag is or is not included)
SumInfl	R(ReKi)			Used to calculate the spatially average induced velocity

- ***EIOutParams***: It contains blade element output information: invocation to Module **Precision** [see DoublePrec.f90 ], 3 real variables, 17 real dynamic arrays, 4 integer(4) variables, and 4 dynamic integer(4) arrays; the user selects which elements need to have info printed out.

Name	Type	Value	Parameter/ Initialized	Description
AAA	( : )	R(ReKi)		Induction factor <i>A</i> for blade 1 elements
AAP	( : )	R(ReKi)		Induction factor <i>AP</i> for blade 1 elements
ALF	( : )	R(ReKi)		ALPHA for blade 1 elements
CDD	( : )	R(ReKi)		CD for blade 1 elements
CLL	( : )	R(ReKi)		CL for blade 1 elements
CMM	( : )	R(ReKi)		CM for blade 1 elements
CNN	( : )	R(ReKi)		CN for blade 1 elements
CTT	( : )	R(ReKi)		CT for blade 1 elements
DFNSAV	( : )	R(ReKi)		Element force normal to plane of rotation for blade 1 elements
DFTSAV	( : )	R(ReKi)		Element force tangent to plane of rotation for blade 1 elements
DynPres	( : )	R(ReKi)		Dynamic Pressure for blade 1 elements
PITSAV	( : )	R(ReKi)		Pitch angle [deg] for blade 1 elements
PMM	( : )	R(ReKi)		Torque for blade 1 elements
ReyNum	( : )	R(ReKi)		Element Reynolds Number
SaveVX	( :, : )	R(ReKi)		The velocity in the x direction at requested element, on each blade
SaveVY	( :, : )	R(ReKi)		The velocity in the y direction at requested element, on each blade
SaveVZ	( :, : )	R(ReKi)		The velocity in the z direction at requested element, on each blade
VXSAV		R(ReKi)		Wind Velocity along global X for last blade element of blade 1
VYSAV		R(ReKi)		Wind Velocity along global Y for last blade element of blade 1

VZSAV	R(ReKi)			Wind Velocity along global Z for last blade element of blade 1
WndElPrList (:)	I(4)			Array(NElm) that for each blade element stores 0 if no printing requested, or a number indicating where the wind element is within the list of wind elements to be printed
WndElPrNum (:)	I(4)			Array(NumWndElOut) with indices of the blade wind elements to print
ElPrList (:)	I(4)			Array(NElm) that for each blade element stores 0 if no printing requested, or a number indicating where the blade element is within the list of elements to be printed
ElPrNum (:)	I(4)			Array(NumElOut) that has the indices of the blade elements to print
NumWndElOut	I(4)			Number of wind elements to print
NumElOut	I(4)			Number of blade elements to print
UnElem	I(4)	94	Initialized	The unit for element ".elm" file
UnWndOut	I(4)	96	Initialized	The unit for wind output at each element on each blade

- Airfoil*: It contains airfoil information: invocation to Module *Precision* [see DoublePrec.f90 ], 2 real variables, 5 real dynamic arrays, 2 integer(4) variables, 3 dynamic integer(4) arrays, 1 integer(4) parameter, and 1 character(1024) dynamic array:

Name	Type	Value	Parameter/ Initialized	Description
AL ( :, : )	R(ReKi)			Array(NumFoil ,NLIFT(I)): Tables of AOAs
CD ( :, :, : )	R(ReKi)			Array(NumFoil , NLIFT(I), NTables(I)): Table of drag coefficients
CL ( :, :, : )	R(ReKi)			Array(NumFoil , NLIFT(I), NTables(I)): Table of lift coefficients
CM ( :, :, : )	R(ReKi)			Array(NumFoil , NLIFT(I), NTables(I)): Table of pitching moment coefficients

<i>MulTabMet( ; :)</i>	R(ReKi)			( <i>NumFoil,Ntables(I)</i> ) Table ID parameters (e.g. Reynolds numbers) per airfoil file
<i>MulTabLoc</i>	R(ReKi)	0.0	Initialized	It is the ‘current’ value of the Table ID to work on interpolations
<i>PMC</i>	R(ReKi)			Moment coefficient (1/4-chord)
<i>NFOIL ( : )</i>	I(4)			Array( <i>NumFoil</i> ): Indices of the airfoil data file used for each element
<i>NLIFT ( : )</i>	I(4)			Array( <i>NumFoil</i> ): Number of aerodata points in each airfoil file
<i>NTables ( : )</i>	I(4)			Array( <i>NumFoil</i> ): number of airfoil data tables(Reynolds numbers)
<i>NumCL</i>	I(4)			Maximum number of aerodata points in all airfoil files {=max(NFoil(:))}
<i>NumFoil</i>	I(4)			number of different airfoil files used
<i>MAXTABLE</i>	I(4)	10	PARAMETER	Max number of airfoil tables
<i>FOILNM ( : )</i>	C(1024)			Array( <i>NumFoil</i> ): names of the data files that contain airfoil data

- *Bedoes*: It contains Bedoes Dynamic stall info; invocation to Module *Precision* [see DoublePrec.f90 ], 16 real variables, 50 real(ReKi) dynamic arrays, 2 logical variables, 2 dynamic logical arrays:

Name	Type	Value	Parameter/ Initialized	Description
<i>AS</i>	R(ReKi)			Speed of sound for Mach number calculation
<i>AOD ( ; : )</i>	R(ReKi)			( <i>NumFoil, Ntables (Airfoil)</i> ) Min Drag AOA
<i>AOL ( ; : )</i>	R(ReKi)			( <i>NumFoil, Ntables (Airfoil)</i> ) Zero-Lift AOA
<i>CDO ( ; : )</i>	R(ReKi)			( <i>NumFoil, Ntables (Airfoil)</i> ) Min Drag Coefficient
<i>CNA ( ; : )</i>	R(ReKi)			( <i>NumFoil, Ntables (Airfoil)</i> ) CN slope
<i>CNS ( ; : )</i>	R(ReKi)			( <i>NumFoil, Ntables (Airfoil)</i> ) Upper Stall CN
<i>CNSL ( ; : )</i>	R(ReKi)			( <i>NumFoil, Ntables (Airfoil)</i> ) Lower Stall CN

<i>TF</i>	R(ReKi)	Time constant applied to location of the separation point
<i>TP</i>	R(ReKi)	Time constant for pressure lag at LE
<i>TV</i>	R(ReKi)	Time constant for strength of shed vortex
<i>TVL</i>	R(ReKi)	Non-dimensional time of transit for the vortex moving across the airfoil surface
<i>AN</i>	R(ReKi)	AOA
<i>ANE ( ; : )</i>	R(ReKi)	( <i>Nelm(Element)</i> , <i>Nb (Blade)</i> ) Modified <i>AN</i> to account for $ AN  > \pi/2$
<i>ANE1 ( ; : )</i>	R(ReKi)	Previous time step value of <i>ANE</i>
<i>AFE ( ; : )</i>	R(ReKi)	<i>Nelm(Element), Nb (Blade)</i> Effective AOA to account for LE pressure gradient lags, and determine TE separation point as well (modified <i>AFF (SEPAR(AeroSubs))</i> to account for $ AFF  > \pi/2$ )
<i>AFE1 ( ; : )</i>	R(ReKi)	Previous time step value of <i>AFE</i>
<i>ADOT ( ; : )</i>	R(ReKi)	( <i>Nelm(Element), Nb (Blade)</i> ) Time derivative Of <i>ANE</i>
<i>ADOT1 ( ; : )</i>	R(ReKi)	Previous time step value of <i>ADOT</i>
<i>DS</i>	R(ReKi)	Infinitesimal delta-s, where s is the distance traveled by the airfoil is semi-chords: $2/c^*U^*dt$
<i>CNCP</i>	R(ReKi)	CN due to circulatory part for attached/potential flow
<i>CNPOT ( ; : )</i>	R(ReKi)	( <i>Nelm(Element), Nb (Blade)</i> ) Total CN (circ.+non-circ.) for attached/potential flow
<i>CNPOT1 ( ; : )</i>	R(ReKi)	<i>CNPOT</i> at previous time step
<i>CNP ( ; : )</i>	R(ReKi)	<i>Cn'</i> , modified <i>Cn</i> to account for lags in pressure gradient response at the LE
<i>CNP1 ( ; : )</i>	R(ReKi)	Previous time step value of <i>CNP</i>
<i>CNPD ( ; : )</i>	R(ReKi)	$\Delta Cn'$ , i.e. $\Delta CNP$
<i>CNPD1 ( ; : )</i>	R(ReKi)	Previous time step value of <i>CNPD</i>
<i>CNIQ</i>	R(ReKi)	Non-circulatory component of <i>Cn</i> in

		attached/potential flow
<i>CNV</i> ( :, : ) R(ReKi)		( <i>Nelm(Element)</i> , <i>Nb (Blade)</i> ) ,Contribution to Cn from LE vortex
<i>OLDCNV</i> ( :, : ) R(ReKi)		Previous time step value of <i>CNV</i>
<i>CVN</i> ( :, : ) R(ReKi)		( <i>Nelm(Element)</i> , <i>Nb (Blade)</i> ), Increment in vortex lift used to calculate <i>CNV</i>
<i>CVN1</i> ( :, : ) R(ReKi)		Previous time step value of <i>CVN</i>
<i>CN</i>	R(ReKi)	Normal-to-chord force coefficient
<i>CC</i>	R(ReKi)	Along-chord force coefficient (TE to LE)
<i>CMI</i>	R(ReKi)	Non-circulatory component of Cm in response to step-change in AOA
<i>CMQ</i>	R(ReKi)	Non-circulatory component of Cm in response to step-change in pitch-rate
<i>DFAFE</i> ( :, : ) R(ReKi)		( <i>Nelm(Element)</i> , <i>Nb (Blade)</i> ), Deficiency function associated with a lagged AFE to compute Cm under TE-separated conditions
<i>DFAFE1</i> ( :, : ) R(ReKi)		Previous time step value of <i>DFAFE</i>
<i>DF</i> ( :, : ) R(ReKi)		( <i>Nelm(Element)</i> , <i>Nb (Blade)</i> ) Deficiency function for separation point f
<i>OLDDF</i> ( :, : ) R(ReKi)		( <i>Nelm(Element)</i> , <i>Nb (Blade)</i> ) Previous time step value of <i>DF</i>
<i>DFC</i> ( :, : ) R(ReKi)		Deficiency function for separation point fc
<i>OLDDFC</i> ( :, : ) R(ReKi)		Previous time step value of <i>DFC</i>
<i>DN</i> ( :, : ) R(ReKi)		( <i>Nelm(Element)</i> , <i>Nb (Blade)</i> ) $K_{\alpha}'$ , i.e. Deficiency Function for non-circulatory Cn in response to step-change in AOA
<i>OLDDN</i> ( :, : ) R(ReKi)		Previous time step value of <i>DN</i>
<i>DQ</i> ( :, : ) R(ReKi)		( <i>Nelm(Element)</i> , <i>Nb (Blade)</i> ) $K_q'$ , Deficiency Function for non-circulatory Cn in response to step-change in q (pitch-rate)
<i>OLDDQ</i> ( :, : ) R(ReKi)		Previous time step value of <i>DQ</i>
<i>DQP</i> ( :, : ) R(ReKi)		( <i>Nelm(Element)</i> , <i>Nb (Blade)</i> ) $K_q''$ Deficiency

		function used in the calculation of CMQ
<i>DQP1</i> ( :, : ) R(ReKi)		Previous time-step value of DQP
<i>DPP</i> ( :, : ) R(ReKi)		( <i>Nelm(Element)</i> , <i>Nb (Blade)</i> ) Deficiency function associated with Cn', to account for lag in LE pressure gradient and LE separation
<i>OLDDPP</i> ( :, : ) R(ReKi)		Previous time-step value of <i>DPP</i>
<i>XN</i> ( :, : ) R(ReKi)		1 <sup>st</sup> deficiency function for AOA at ¾-chord Attached flow response
<i>YN</i> ( :, : ) R(ReKi)		2 <sup>nd</sup> deficiency function for AOA at ¾-chord Attached flow response
<i>OLDXN</i> ( :, : ) R(ReKi)		Previous time-step value of <i>XN</i>
<i>OLDYN</i> ( :, : ) R(ReKi)		Previous time-step value of <i>YN</i>
<i>TAU</i> ( :, : ) R(ReKi)		Non-dimensional time variable tracking the vortex convection along the airfoil
<i>OLDTAU</i> ( :, : ) R(ReKi)		Previous time-step value for <i>TAU</i>
<i>QX</i> ( :, : ) R(ReKi)		( <i>Nelm(Element)</i> , <i>Nb (Blade)</i> ) Time derivative of non-dimensional pitching rate
<i>QX1</i> ( :, : ) R(ReKi)		Previous time-step value of <i>QX</i>
<i>FTB</i> ( :, :, : ) R(ReKi)		( <i>Numfoil</i> , <i>Nlift(I)</i> , <i>Ntables(I)</i> ) ( <i>Airfoil</i> ) Tabulated f values for airfoil Cn
<i>FTBC</i> ( :, :, : ) R(ReKi)		( <i>Numfoil</i> , <i>Nlift(I)</i> , <i>Ntables(I)</i> ) ( <i>Airfoil</i> ) Tabulated f values for airfoil Cc
<i>FSP</i> ( :, : ) R(ReKi)		f' for reconstructing Cn (f' is an effective f to account for LE pressure gradient)
<i>FSP1</i> ( :, : ) R(ReKi)		Previous step value of <i>FSP</i>
<i>FSPC</i> ( :, : ) R(ReKi)		fc'=f' for reconstructing Cc (fc' is an effective fc to account for LE pressure gradient)
<i>FSPC1</i> ( :, : ) R(ReKi)		Previous step value of <i>FSPC</i>
<i>FP</i> R(ReKi)		Lagged f', f'' (see <u>SEPAR (AeroSubs)</u> )
<i>FPC</i> R(ReKi)		Lagged fc', fc'' (see <u>SEPAR (AeroSubs)</u> )
<i>FK</i> R(ReKi)		0.25*(1+SQRT(f'))^2

<i>BEDSEP ( ; ; )</i>	L	( <i>Nelm(Element)</i> , <i>Nb (Blade)</i> ) LE Separation Flag
<i>OLDSEP ( ; ; )</i>	L	Previous time-step value of <i>BEDSEP</i>
<i>SHIFT</i>	L	True if AOA is decreasing, vortex decay is faster
<i>VOR</i>	L	Whether or not vortex lift is allowed to accumulate or just decay

- *DynInflow*: It contains Dynamic Inflow Info; invocation to Module *Precision* [see DoublePrec.f90], 22 real(ReKi) variables/arrays, 2 real(ReKi) dynamic arrays, 5 integer(4) variables/arrays, 2 integer(4) parameters:

Name	Type	Value	Parameter/ Initialized	Description
<i>MAXINFL</i>	I(4)	6	PARAMETER	Maximum number of radial shapes
<i>MAXINFL0</i>	I(4)	2	PARAMETER	Maximum number of radial states associated with 0-th harmonic mode in azimuth
<i>MminR ( maxInfl, maxInfl )</i>	I(4)			Matrix (i,j) formed with the min( <i>MRvector(i)</i> , <i>MRvector(j)</i> )
<i>MminusR ( maxInfl, maxInfl )</i>	I(4)			Matrix (i,j) formed with the  ( <i>MRvector(i)</i> - <i>MRvector(j)</i>
<i>MplusR ( maxInfl, maxInfl )</i>	I(4)			Matrix (i,j) formed with the [( <i>MRvector(i)</i> + <i>MRvector(j)</i> )]
<i>MRvector ( maxInfl )</i>	I(4)			[0, 0, 1, 1, 2, 3]; indices of the azimuthal modes considered so far in AeroDyn, one per state
<i>NJvector ( maxInfl )</i>	I(4)			[1, 3, 2, 4, 3, ]; indices of the azimuthal modes considered so far in AeroDyn, one per state and corresponding to the <i>MRVector</i> values
<i>RMC_SAVE( : , : , : )</i>	R(ReKi)			( <i>Nb(Blade)</i> , <i>Nelm(Element)</i> , <i>Maxinfl</i> ) Mode-moments of the BE force, i.e: BE force times the radial shape functions for all of the

		modes, for the cosine terms
<i>RMS_SAVE( :, :, :, : )</i>	R(ReKi)	As <i>RMC_SAVE</i> for sine terms
<i>DT0</i>	R(ReKi)	Non-dimensional time interval
<i>GAMMA (maxInfl, maxInfl )</i>	R(ReKi)	Matrix formed with values of $\Gamma_{ln}^{rm}$
<i>xAlpha(maxInfl )</i>	R(ReKi)	Inflow States relative to cosines
<i>xBeta ( maxInfl0+1 : maxInfl )</i>	R(ReKi)	Inflow States relative to sines
<i>old_Alph (maxInfl )</i>	R(ReKi)	Previous Time-step values of <i>xAlpha</i>
<i>old_Beta( maxInfl0+1:maxInfl)</i>	R(ReKi)	Previous Time-step values of <i>xBeta</i>
<i>dAlph_dt( maxInfl, 4)</i>	R(ReKi)	Time-derivative of Alphas (Inflowstates) + 3 previous values
<i>dBeta_dt(maxInfl0+1 :maxInfl,4)</i>	R(ReKi)	Time-derivative of Betas (Inflowstates) + 3 previous values
<i>XLAMBDA_M</i>	R(ReKi)	Induced velocity normal to rotor plane
<i>old_LmdM</i>	R(ReKi)	Previous time step value of <i>XLAMBDA_M</i>
<i>xKai</i>	R(ReKi)	$\text{Tan}(\chi/2)$
<i>oldKai</i>	R(ReKi)	Previous Time-step values of <i>xKai</i>
<i>PhiLqC (maxInfl )</i>	R(ReKi)	Value of integrals of BE forces times radial shape functions, needed to calculate $\tau_n^{mc}$
<i>PhiLqS ( maxInfl0+1 : maxInfl )</i>	R(ReKi)	As <i>PhiLqC</i> , but to calculate $\tau_n^{ms}$ $=\pi * \rho (\text{Wind}) * \text{TipSpeed}^{**2} * R$ , normalization factor for $\tau_n^{mc} \tau_n^{ms}$
<i>TipSpeed</i>	R(ReKi)	Tip-speed
<i>totalInf</i>	R(ReKi)	Total wind velocity normal to rotor plane including induction
<i>Vparam</i>	R(ReKi)	The mass-flow parameter
<i>Vtotal</i>	R(ReKi)	Total wind velocity magnitude at the rotor (induction included)

$xLcos$ ( $maxInfl, maxInfl$ )	R(ReKi)	This is $[\tilde{L}^c]^{-1}$ (or $[\tilde{L}^c]$ ) (see Theory)
$xLsin$ ( $maxInfl0+1 : maxInfl,$ $maxInfl0+1 : maxInfl$ )	R(ReKi)	This is $[\tilde{L}^s]^{-1}$ (or $[\tilde{L}^s]$ ) (see Theory)
$xMinv$ ( $maxInfl$ )	R(ReKi)	Inverse of matrix [M], stored as an array

- ***TwrProps***: It contains tower aero-info; invocation to Module **Precision** [see DoublePrec.f90 ], 4 real(ReKi) variables/arrays, 7 real(ReKi) dynamic arrays, 1 integer dynamic array, 3 integer variables/arrays, 3 logical variables, 1 string:

Name	Type	Value	Parameter/ Initialized	Description
$TwrHtFr ( : )$	R(ReKi)			Tower Station Height Fraction
$TwrWid ( : )$	R(ReKi)			Tower Station Width
$TwrCD ( :, : )$	R(ReKi)			Tower Station CD per Reynolds number
$TwrRe ( : )$	R(ReKi)			Reynolds Numbers
$VTwr(3)$	R(ReKi)			NOT USED
<i>Tower_Wake_Constant</i>	R(ReKi)			Constant for tower wake model = 0 full potential flow = 0.1 model of Bak et al.
<i>SHADHWID</i>	R(ReKi)			Shadow width from user input
<i>TSHADC1</i>	Re(reKi)			<i>AD_GetInput(AeroSubs)</i> sets it = $ShadHWid / \sqrt{ABS(T_Shad_Refpt)}$
<i>TSHADC2</i>	Re(reKi)			<i>AD_GetInput(AeroSubs)</i> sets it = $TwrShad / \sqrt{ABS(T_Shad_Refpt)}$
<i>TWRSHAD</i>	Re(reKi)			Tower shadow deficit from user input
<i>T_Shad_Refpt</i>	Re(reKi)			This was a local variable -- with new tower influence, it should be removed
<i>NTwrCDCol</i> (:)	I			The tower CD column that represents the CD for that particular tower height
<i>NTwrHt</i>	I			The number of tower height rows in the table
<i>NTwrRe</i>	I			The number of tower Re entry rows in the

table

<i>NTwrCD</i>	I	The number of tower CD columns in the table
<i>TwrPotent</i>	L	Tower potential flow calculation
<i>TwrShadow</i>	L	Tower Shadow calculation
<i>PJM_Version</i>	L .FALSE.	Whether new(T) or old(F) tower model is used
<i>TwrFile</i>	C(1024)	Name of the tower properties input file

- *Wind*: It contains tower aero-info; invocation to Module *Precision* [see DoublePrec.f90 ], 8 real(ReKi) variables:

Name	Type	Value	Parameter/ Initialized	Description
<i>ANGFLW</i>	R(ReKi)			Angle between wind velocity and rotor plane
<i>CDEL</i>	R(ReKi)			$\text{Cos}(\text{angle between horizontal rotor axis } y \text{ and in-plane wind velocity})$
<i>KinVisc</i>	R(ReKi)			Kin. Viscosity
<i>RHO</i>	R(ReKi)			Air Density
<i>SDEL</i>	R(ReKi)			$\text{Sin}(\text{angle between horizontal rotor axis } y \text{ and in-plane wind velocity})$
<i>VROTORX</i>	R(ReKi)			Wind velocity component along hub x-axis (rotational).
<i>VROTORY</i>	R(ReKi)			Wind velocity component along hub y-axis+yawing effects
<i>VROTORZ</i>	Re(reKi)			Wind velocity component along hub z-axis(upward). Hub axes not rotating.

- *ErrCount*: It contains error counters; 2 intger(4) variables:

Name	Type	Value	Parameter/ Initialized	Description
<i>NumErr</i>	I(4)			Number of errors
<i>NumWarn</i>	I(4)			Number of warnings

## APPENDIX E. AEROGENSUBS MODULE

### GENERAL ORGANIZATION

GenSubs.f90 contains the module *AeroGenSubs*.

It contains the SubRoutines: *AllocArrays*, *ElemOpen*, *ElemOut*

### EXTERNAL (INVOKED) MODULES

- *NWTC\_LIBRARY*

## E1. ELEMOPEN(ELEMFILE)

Subroutine that opens the element ouput file and writes tab-spaced column headings. Called by **AD\_Init (AeroDyn)**.

### EXTERNAL (INVOKED) MODULES

- *Blade* [see AeroMods.f90]
- *EIOutParams* [see AeroMods.f90]
- *Element* [see AeroMods.f90]
- *Switch*[see AeroMods.f90]

### INPUT

*ElemFile*: character(\*); File name

### LOCAL VARIABLES

A few counters and strings.

Name	Type	Value	Parameter/ Initialized	Description
<i>JE</i>	I(4)			Counter for blade element
<i>JB</i>	I(4)			Counter for blade
<i>Dst_Unit</i>	C(2)			Distance Unit String
<i>Frc_Unit</i>	C(2)			Force Unit String
<i>Frmt</i>	C(140)			Format String
<i>Prs_Unit</i>	C(3)			Pressure Unit String

### STEPS

1. IF *NumWndEIOut (EIOutParams)* > 0 THEN
  - a. Open output wind *UnWndOut* file unit (*EIOutParams*)
  - b. Write program version, and current date and time *CurDate()*, *CurTime()* (*NWTC\_IO* within *NWTC\_Library*)
2. IF *ELEMPRN (Switch)* THEN
  - a. Open output element file unit *UnElem* (*EIOutParams*)
  - b. Write program version, and current date and time *CurDate()*, *CurTime()*
3. Set unit Labels *Dst\_Unit*, *Frc\_Unit* , *Prs\_Unit* to appropriate strings ('m', 'N', 'Pa') (here there is still a possibility of English units, it should be removed?)
4. Set *Frmt* format string for following writeouts; Note that a slick way to use WRITE is used to adjust the format based on the number of elements to be printed out
5. Determine whether *PMOMENT==.TRUE.* and write the headings (in case include the appropriate string for Pitching moment) for the element output information ('Time', 'VX', 'VY'...'ForcN',

'ForcT', etc) including the element numbers along the column headings. For 'VX', 'VY', 'VZ' the element number is selected as *NELM (Element)* i.e. the last element number

6. Add units to the headings
7. IF *NumWndElOut (ElOutParams)* > 0 THEN on *UnWndOut* file unit
  - a. Set *Frmt* format string for following writeouts; Note that a slick way to use WRITE is used to adjust the format based on the number of elements to be printed out
  - b. Write the headings for the element wind output information ('Time', 'VX', 'VY', 'VZ') including the blade number and element numbers along the column headings
  - c. Add unit labels

## **OUTPUT**

It will write headings for the information such as time stamp, relative velocity, AOA, dynamic pressure, Cl, Cd, Cn, Ct, pitch, a, a', Reynolds, etc. on element output file, and VX, VY, VZ on wind element output file ".elm". Additionally similar headings ('Time', 'VX', 'VY', 'VZ') on the wind element output file.

## E2. ELEMOUT()

Subroutine that writes output values for the desired elements. Externally Called.

### EXTERNAL (INVOKED) MODULES

- *Blade* [see AeroMods.f90]
- *EIOutParams* [see AeroMods.f90]
- *AeroTime* [see AeroMods.f90]
- *Switch* [see AeroMods.f90]

### INPUT

N/A

### LOCAL VARIABLES

A few counters and strings.

Name	Type	Value	Parameter/ Initialized	Description
<i>JE</i>	I(4)			Counter for blade element
<i>JB</i>	I(4)			Counter for blade
<i>Frmt</i>	C(140)			Format String

### STEPS

1. IF *ELEMPRN* (*Switch*) THEN on *UnElem* (*EIOutParams*) output file unit:
  - a. Write TIME (*AeroTime*), VXSAV, VYSAV, VZSAV, (*EIOutParams*), ALF (JE), DynPres(JE), CLL(JE), CDD(JE), CNN(JE), CTT(JE), CMM(JE), PITSAV(JE), AAA(JE), AAP(JE), DFNSAV(JE), DFTSAV(JE), PMM(JE) (only IF PMOMENT (*Switch*)), ReyNum(JE) with JE= 1, *NumElOut* (all from *EIOutParams*)
2. IF *NumWndElOut* (*EIOutParams*) > 0 THEN on *UnWndOut* file unit
  - a. Write TIME (*AeroTime*), *SaveVX*(JE, JB), *SaveVY*(JE, JB) *SaveVZ*(JE, JB) (all from *EIOutParams*) with JE= 1, *NumElOut* and JB =1, NB (*Blade*)

### OUTPUT

It will write onto the element “.elm” output file, information such as time stamp, wind velocity (VXSAV, VYSAV, VZSAV for the blade last element), AOA, dynamic pressure, Cl, Cd, Cn, Ct, pitch, a, a', Reynolds, etc. for the elements selected in the input file. It will write onto the wind element output file the wind velocity (*SaveVX*(JE, JB), *SaveVY*(JE, JB) *SaveVZ*(JE, JB)) for each blade and requested element.

### E3. ALLOCARRAYS(ARG)

Subroutine that allocates dynamic arrays. Called by : AD GetInput, READFL (*AeroSubs*)

#### EXTERNAL (INVOKED) MODULES

- *Airfoil* [see AeroMods.f90]
- *Bedoes* [see AeroMods.f90]
- *Blade* [see AeroMods.f90]
- *DynInflow* [see AeroMods.f90]
- *Element* [see AeroMods.f90]
- *ElOutParams* [see AeroMods.f90]
- *Switch* [see AeroMods.f90]

#### INPUT

Name	Type	INTENT	Description
<i>Arg</i>	C(*)	IN	Generic String to direct the subroutine

#### OUTPUT

When called with *Arg*=’Element’, it will allocate (& possibly initialize) arrays from module **ElOutParams** (*ElPrList(:)=0, WndElPrList(:)=0*), **Element** (*A(:, :)=0.0, AP, RELM, TWIST, TLCNST=99.0, HLCNST=99.0*), and if *DSTALL* then from **Bedoes** (*ADOT(:, :)=0.0, ADOT1, AFE(:, :)=0.0, AFE1, ANE(:, :)=0.0, ANE1, AOC, AOL, BEDSEP(:, :)=.FALSE., CDO, CAN, CNP(:, :)=0.0, CNP1, CNPD(:, :)=0.0, CNPD1, CNPOT(:, :)=0.0, CNPOT1, CNS, CNSL, CNV(:, :)=0.0, CVN(:, :)=0.0, CVN1, DF(:, :)=0.0, DFAFE(:, :)=0.0, DFAFE1, DFC(:, :)=0.0, DN(:, :)=0.0, DPP(:, :)=0.0, DQ(:, :)=0.0, DQP(:, :)=0.0, DQP1, FSP(:, :)=0.0, FSP1, FSPC(:, :)=0.0, FSPC1, OLDCNV, OLDDF, OLDDFC, OLDDN, OLDDP, OLDDQ, OLDTAU, OLDXN, OLDYN, OLDSEP, QX(:, :)=0.0, QX1, TAU(:, :)=0.0, XN(:, :)=0.0, YN(:, :)=0.0*), **Blade** (*C, DR*) , **Airfoil** (*NFOIL, NLIFT, NTABLES*) , **DynInflow** (*RMC\_SAVE=0.0, RMS\_SAVE=0.0*).

When called with *Arg*=’ElPrint’ (and *NumElOut>0*), it will allocate arrays from module **ElOutParams** (*AAA, AAP, ALF, CDD, CLL, CMM, CNN, CTT, DFNSAV, DFTSAV, DynPres, PMM, PITSAV, ReyNum, ElPrNum(:)=0*); and if *NumWndElOut>0* from module **ElOutParams** (*WndElPrNum(:)=0, SaveVX, SaveVY, SaveVz*).

When called with *Arg*=’Aerodata’, it will allocate arrays from module **Airfoil** (*AL, Cd, Cl, Cm, MulTabMet*) and if *DSTALL* from **Bedoes** (*FTB, FTBC*).

## APPENDIX F. SHARED TYPES MODULE

### GENERAL ORGANIZATION

SharedTypes.f90 contains the module *SharedTypes*, which includes a series of new derived types (structure types).

### EXTERNAL MODULES

- *NWTC\_Library*

### PUBLIC (LOCALLY DECLARED) DATA

- **Marker:** NEW TYPE made up of 4 real arrays:

Name	Type	Value	Parameter/ Initialized	Description
<i>Position(3)</i>	Re(ReKi)			X,Y,Z in FF0
<i>Orientation(3,3)</i>	Re(ReKi)			Direction Cosine Matrix
<i>TranslationVel(3)</i>	Re(ReKi)			3 components of translational velocity
<i>RotationVel(3)</i>	Re(ReKi)			3 components of rotational velocity

- **Load:** NEW TYPE made up of 2 real arrays:

Name	Type	Value	Parameter/ Initialized	Description
<i>Force(3)</i>	Re(ReKi)			3-components of force
<i>Moment(3)</i>	Re(ReKi)			3-components of moment

- **AllAeroMarkers:** NEW TYPE made up of 6 **Marker** dynamic arrays:

Name	Type	Value	Parameter/ Initialized	Description
Blade(:, :)	<b>Marker</b>			(Nelm ( <i>Element</i> ), Nb ( <i>Blade</i> )) ( <i>Aeromods</i> [.f90])
Hub(:)	<b>Marker</b>			Not sure why array
RotorFurl(:)	<b>Marker</b>			Not sure why array
Nacelle(:)	<b>Marker</b>			Not sure why array
Tower(:)	<b>Marker</b>			Not sure why array, perhaps vertical stations?
Tail(:)	<b>Marker</b>			Not sure why array

- **AllAeroLoads:** NEW TYPE made up of 6 Load dynamic arrays:

Name	Type	Value	Parameter/ Initialized	Description
Blade(:, :)	Load			(Nelm ( <i>Element</i> ), Nb ( <i>Blade</i> ))
Hub(:)	Load			Not sure why array
RotorFurl(:)	Load			Not sure why array
Nacelle(:)	Load			Not sure why array
Tower(:)	Load			Not sure why array, perhaps vertical stations?
Tail(:)	Load			Not sure why array

- **ProgDesc:** NEW TYPE made up of 2 strings:

Name	Type	Value	Parameter/ Initialized	Description
Name	C(20)			Prg. name
Ver	C(99)			Prg. version

◦

- **AeroConfig:** NEW TYPE made up of 1 dynamic array of type Marker, 1 real variable, and 7 variables of type Marker:

Name	Type	Value	Parameter/ Initialized	Description
Hub	Marker			Self-explanatory
RotorFurl	Marker			"
Nacelle	Marker			"
TailFin	Marker			"
Tower	Marker			"
Substructure	Marker			"
Foundation	Marker			"
Blade(:)	Marker			(Nb( <i>Blade</i> ))
BladeLength	Re(ReKi)			Self-explanatory

## APPENDIX G. INFLOWWIND MODULE

### GENERAL ORGANIZATION

InflowWindMod.f90 contains the module *InflowWind*, which includes a series of new variables, one new type, and a series of subroutines and functions.

This module is used to read and process the (undisturbed) inflow winds. It must be initialized using *WindInf\_Init()* with the name of the file, the file type, and possibly reference height and width (depending on the type of wind file being used). This module calls appropriate routines in the wind modules so that the type of wind becomes seamless to the user. *WindInf\_Terminate()* should be called when the program has finished.

Data are assumed to be in units of meters and seconds. Z is measured from the ground (NOT the hub!).

### EXTERNAL MODULES

- *NWTC\_Library*
- *SharedInflowDefns* [see SharedInflowDefs.f90]
- *FFWind* [see FFWind.f90]
- *HHWind* [see HHWind.f90]
- *FDWind* [see FDWind.f90]
- *CTWind* [see CTWind.f90]
- *UserWind* [see UserWind.f90]

### PRIVATE (LOCALLY DECLARED) DATA

Name	Type	Value	Parameter/ Initialized	Description
<i>WindType</i>	I	0	Initialized +Save	Wind Type Flag
<i>UnWind</i>	I	91	Initialized	Unit number for wind inflow file
<i>CT_Flag</i>	L	.FALSE.	Initialized + Save	Determines if coherent turbulence is used

### PUBLIC (LOCALLY DECLARED) DATA

- **InflInitInfo:** PUBLIC NEW TYPE made up of 2 real, 1 string, and 1 integer fields:

Name	Type	Value	Parameter/ Initialized	Description
<i>WindFileName</i>	C(1024)			Name[path] of the wind file
<i>WindFileType</i>	I			Code for wind file type
<i>ReferenceHeight</i>	Re(ReKi)			Reference height for HH and/or 4D winds (was hub height), in meters
<i>Width</i>	Re(ReKi)			Width of the HH file (was 2*R), in meters

- New PUBLIC Routines and Functions

Name	Description
<u>WindInf_Init</u>	Initialization subroutine
<u>WindInf_GetVelocity</u>	Function to get wind speed at point in space and time
<u>WindInf_GetMean</u>	Function to get the mean wind speed at a point in space
<u>WindInf_GetStdDev</u>	Function to calculate standard deviation at a point in space
<u>WindInf_GetTI</u>	Function to get TI at a point in space
<u>WindInf_Terminate</u>	Subroutine to clean up
<u>WindInf_ADhack_diskVel</u>	It keeps old AeroDyn functionality--remove soon
<u>WindInf_ADhack_DIcheck</u>	It keeps old AeroDyn functionality--remove soon
<u>WindInf_LinearizePerturbation</u>	It is used for linearization; should be modified

- New LOCAL Routines and Functions

Name	Description
<u>GetWindType</u>	It returns a code for the wind file type based on filename extension

## G1. WINDINF\_INIT(FILEINFO, ERRSTAT)

Subroutine that opens and reads wind files allocating space for necessary variables. Called by [AD\\_Init](#) (*Aerodyn*).

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>FileInfo</i>	<i>InflInitInfo (InflowWind)</i>	IN	Main variable
<i>ErrStat</i>	I	OUT	If $\neq 0$ error encountered

Additional input is given through invoked module variables and parent module variables.

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>HHInitInfo</i>	<i>HH_Info (HHWind)</i>			It is used with <u><a href="#">HH_Init</a></u> ( <i>HHWind</i> )
<i>BackGrndValues</i>	<i>CT_Backgr (CTWind)</i>			Outout of <u><a href="#">CT_Init</a></u> ( <i>CTWind</i> )
<i>Height</i>	R(ReKi)			Aux variable for hub-height and FF wind file
<i>HalfWidth</i>	R(ReKi)			Aux variable for GridWidth and FF wind file
<i>FileName</i>	C(1024)			Wind File Name

#### STEPS

1. IF *WindType* (parent module *InflowWind*) $\neq 0$ , wind flow was already initialized, THEN put *ErrStat*=1 and RETURN;
2. ELSE Set:
  - a. *WindType*=*FileInfo*%*WindFileType* (Input came from [AD\\_Init](#) in *Aerodyn*)
  - b. *FileName* = *FileInfo*%*WindFileName* (Input came from [AD\\_Init](#) in *Aerodyn*)
3. IF *WindType* == *DEFAULT\_Wind* (*SharedInflowDefns*), THEN set *WindType* = [GetWindType](#)(*FileName*, *ErrStat*) (*InflowWind*); i.e. find the wind type if not set by user a priori, however AD\_Init puts the type equal to the default value, so unclear why there is a check to be done here (minor bug?).
4. IF *WindType* ==*CTP\_Wind* (*SharedInflowDefns*) then: Coherent Turbulent File
  - a. call [CT\\_Init](#)(*UnWind*, *FileName*, *BackGrndValues*, *ErrStat*) (*CTWind*) :Initialize the *CTWind* module
  - b. IF *ErrStat* $\neq 0$  THEN:
    - i. call [WindInf\\_Terminate](#)(*ErrStat*) (*InflowWind*)

- ii. set *WindType* = 0
- iii. *ErrStat* = 1
- iv. RETURN
- c. Set *FileName* = *BackGrndValues*%*WindFile*
- d. Set *WindType* = *BackGrndValues*%*WindFileType*
- e. Set *CT\_Flag* (*InflowWind*) = *BackGrndValues*%*CoherentStr*
- 5. ELSE set *CT\_Flag* = .FALSE.
- 6. Based on *WindType*
  - a. *HH\_Wind* (*SharedInflowDefns*):
    - i. *HHInitInfo*%*ReferenceHeight* = *FileInfo*%*ReferenceHeight*
    - ii. *HHInitInfo*%*Width* = *FileInfo*%*Width*
    - iii. Call **HH\_Init**(*UnWind*, *FileName*, *HHInitInfo*, *ErrStat*) (*HHWind*)
    - iv. IF *ErrStat*=0 AND *CT\_Flag* (CT file to be used as well) THEN Call **CT\_SetRefVal**(*FileInfo*%*ReferenceHeight*, REAL(0.0, ReKi), *ErrStat*) (*CTWind*)
  - b. *FF\_Wind* (*SharedInflowDefns*):
    - i. Call **FF\_Init**(*UnWind*, *FileName*, *ErrStat*) (*FFWind*)
    - ii. IF *ErrStat*=0 AND *CT\_Flag* THEN:
      - 1. *Height*=**FF\_GetValue**('HubHeight', *ErrStat*) (*FFWind*)
      - 2. IF *ErrStat*<>0 THEN reset *Height* =*FileInfo*%*ReferenceHeight*
      - 3. *HalfWidth*=0.5\***FF\_GetValue**('GridWidth', *ErrStat*) (*FFWind*)
      - 4. IF *ErrStat*<>0 THEN *HalfWidth*=0
      - 5. Call **CT\_SetRefVal**(*Height*, *HalfWidth*, *ErrStat*) (*CTWind*)
  - c. *UD\_Wind* (*SharedInflowDefns*):
    - i. Call **UsrWnd\_Init**(*ErrStat*) (*UserWind*)
  - d. *FD\_Wind* (*SharedInflowDefns*):
    - i. Call **FD\_Init**(*UnWind*, *FileName*, *FileInfo*%*ReferenceHeight*, *ErrStat*) (*FDWind*)
  - e. All other cases should lead to an error *ErrStat*=1 and RETURN
- 7. If *ErrStat*<>0:
  - a. Call **WindInf\_Terminate**( *ErrStat* ) (*InflowWind*)
  - b. Set *WindType*=0
  - c. Set *ErrStat*=1

## OUTPUT

*ErrStat*: I; the usual error status flag.

It sets the variables needed to process the wind file data and that are contained in the respective modules (*HHWind*...*UserWind*) and reads in the wind file.

## G2. WINDINF\_GETVELOCITY(TIME, INPUTPOSITION, ERRSTAT)

Function that gets the wind speed at a point in space and time. Called by [AD GetUndisturbedWind](#) (*Aerodyn*) and [AD WindVelocityWithDisturbance](#) (*AeroSubs*).

### EXTERNAL (INVOKED) MODULES

See Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>Time</i>	R(ReKi)	IN	Time
<i>InputPosition(3)</i>	R(ReKi)	IN	X,Y,Z of BE
<i>ErrStat</i>	I	OUT	Error status flag

Additional input is given through invoked module variables

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>CTWindSpeed</i>	<b>InflIntrpOut</b> <i>(SharedInflowDefs)</i>			See <b>InflIntrpOut</b>

#### STEPS

1. Based on *WindType* (parent module *InflowWindMod*):
  - a. *HH\_Wind*: *WindInf\_GetVelocity* = [HHGetWindSpeed](#)( *Time*, *InputPosition*, *ErrStat*) (*HHWind*)
  - b. *FF\_Wind*: *WindInf\_GetVelocity* = [FFGetWindSpeed](#)( *Time*, *InputPosition*, *ErrStat*) (*FFWind*)
  - c. *UD\_Wind*: *WindInf\_GetVelocity* = [UsrWnd GetWindSpeed](#)( *Time*, *InputPosition*, *ErrStat*) (*UserWind*)
  - d. *FD\_Wind*: *WindInf\_GetVelocity* = [FD GetWindSpeed](#)( *Time*, *InputPosition*, *ErrStat*) (*FDWind*)
  - e. Else set *WindInf\_GetVelocity%Velocity*(:) = 0.0 and *ErrStat*=1
2. If *CT\_Flag* THEN add coherent turbulence in case:
  - a. *CTWindSpeed* = [CT GetWindSpeed](#)(*Time*, *InputPosition*, *ErrStat*) (*CTWind*)
  - b. If *ErrStat*=0, *WindInf\_GetVelocity%Velocity*(:) = *WindInf\_GetVelocity%Velocity*(:) + *CTWindSpeed%Velocity*(:),

#### OUTPUT

*WindInf\_GetVelocity* : TYPE(**InflIntrpOut**) (*SharedInflowDefs*)

It returns the wind speed components U-V-W for the time and position requested, it makes use of specific subroutines to calculate the wind speed depending on the type of wind file.

### G3. WINDINF\_GETMEAN(STARTTIME, ENDTIME, DELT\_TIME, INPUTPOSITION, ERRSTAT)

Function that gets the mean wind speed at a point in space and time. Called Externally if at all.

#### EXTERNAL (INVOKED) MODULES

See Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>StartTime</i>	R(ReKi)	IN	Start time for average
<i>EndTime</i>	R(ReKi)	IN	End time for average
<i>Delta_time</i>	R(ReKi)	IN	Delta-time for average
<i>InputPosition(3)</i>	R(ReKi)	IN	X,y,z, position
<i>ErrStat</i>	I	OUT	Error status flag

Additional input is given through invoked module variables

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>Time</i>	R(ReKi)			Time array
<i>SumVel(3)</i>	R(DbKi)			Sum of velocity magnitude array
<i>I</i>	I			counter
<i>Nt</i>	I			Number of time intervals
<i>NewVelocity</i>	<b>InflIntrpOut</b> <i>(SharedInflowDefs)</i>			U,V,W velocities

#### STEPS

1. Set  $Nt = (EndTime - StartTime) / delta\_time$  ; number of time intervals
2. Initialize
  - a.  $SumVel(:) = 0.0$
  - b.  $ErrStat = 0$
3. DO\_LOOP I=1,Nt :
  - a.  $Time = StartTime + (I-1)*delta\_time$  ; not used
  - b.  $NewVelocity = \underline{WindInf\_GetVelocity}(Time, InputPosition, ErrStat)$  (*InflowWind*)
  - c. If  $ErrStat < 0$  THEN :
    - i.  $WindInf\_GetMean(:) = SumVel(:) / REAL(I-1, ReKi)$
    - ii. RETURN

- d. ELSE:
  - i.  $SumVel(:) = SumVel(:) + NewVelocity \% Velocity(:)$
- 4. END\_LOOP
- 5. Set Output:  $WindInf\_GetMean(:) = SumVel(:) / REAL(Nt, ReKi)$

## OUTPUT

$WindInf\_GetMean(3)$ : R(ReKi); Mean U, V, W

It returns the wind speed components U-V-W for the time interval and position requested, it makes use of specific subroutines to calculate the wind speed depending on the type of wind file.

## G4. WINDINF\_GETSTDDEV(STARTTIME,ENDTIME,DELT\_TIME, INPUTPOSITION,ERRSTAT)

Function that gets the wind speed std.dev. at a point in space and time. Called Externally if at all.

### EXTERNAL (INVOKED) MODULES

See Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>StartTime</i>	R(ReKi)	IN	Start time for average
<i>EndTime</i>	R(ReKi)	IN	End time for average
<i>Delta_time</i>	R(ReKi)	IN	Delta-time for average
<i>InputPosition(3)</i>	R(ReKi)	IN	X,y,z, position
<i>ErrStat</i>	I	OUT	Error status flag

Additional input is given through invoked module variables

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>Time</i>	R(ReKi)			Time array
<i>SumAry(3)</i>	R(DbKi)			Auxiliary array for the averages
<i>MeanVel(3)</i>	R(DbKi)			Mean velocity in the interval
<i>Velocity(:,::)</i>	R(ReKi)			(3,Nt) velocity component array
<i>I</i>	I			Counter
<i>Nt</i>	I			Number of time intervals
<i>NewVelocity</i>	<b>InflIntrpOut</b> <i>(SharedInflowDefs)</i>			U,V,W velocities

#### STEPS

1. Initialize: *WindInf\_GetStdDev(:) = 0.0*
2. Set *Nt= (EndTime - StartTime) / delta\_time* ; number of time intervals
3. IF ( *Nt < 2* ) RETURN; 0 std. dev.
4. IF .NOT. ALLOCATED allocate *Velocity(3,Nt)*; in case of *ErrStat>0* warn user and RETURN
5. Initialize: *SumAry(:) = 0.0*
6. DO\_LOOP *I=1,Nt* :
  - a. *Time = StartTime + (I-1)\*delta\_time* ; not used

- b.  $NewVelocity = \text{WindInf\_GetVelocity}(Time, InputPosition, ErrStat)$  (*InflowWind*); RETURN  
in case ErrStat<>0
  - c.  $Velocity(:,I) = NewVelocity \% Velocity(:)$
  - d.  $SumAry(:) = SumAry(:) + NewVelocity \% Velocity(:)$
7. END\_LOOP
8.  $MeanVel(:) = SumAry(:) / \text{REAL}(Nt, ReKi)$ ; this is the mean velocity
9. Reset  $SumAry(:) = 0.0$  to calc std. dev.
10. DO\_LOOP I=1,Nt
- a.  $SumAry(:) = SumAry(:) + (Velocity(:,I) - MeanVel(:))^{**2}$
11. END\_LOOP
12. Set Output:  $WindInf\_GetStdDev(:) = \text{SQRT}(SumAryl(:) / (Nt-1))$

### **OUTPUT**

*WindInf\_GetStdDev(3): R(ReKi); std. devs. For the components U, V, W of the velocity*

It returns the standard deviation of wind speed components U-V-W for the time interval and position requested, it makes use of specific subroutines to calculate the wind speed depending on the type of wind file.

## G5. WINDINF\_GETTI(STARTTIME, ENDTIME, DELT\_TIME, INPUTPOSITION, ERRSTAT)

Function that gets the turbulence intensities. It is a copy of *WindInf GetStdDev (InflowWind)*. Called Externally if at all.

### EXTERNAL (INVOKED) MODULES

See Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>StartTime</i>	R(ReKi)	IN	Start time for average
<i>EndTime</i>	R(ReKi)	IN	End time for average
<i>Delta_time</i>	R(ReKi)	IN	Delta-time for average
<i>InputPosition(3)</i>	R(ReKi)	IN	X,y,z, position
<i>ErrStat</i>	I	OUT	Error status flag

Additional input is given through invoked module variables

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>Time</i>	R(ReKi)			Time array
<i>SumAry(3)</i>	R(DbKi)			Auxiliary array for the averages
<i>MeanVel(3)</i>	R(DbKi)			Mean velocity in the interval
<i>Velocity(:, :, :)</i>	R(ReKi)			(3,Nt) velocity component array
<i>I</i>	I			Counter
<i>Nt</i>	I			Number of time intervals
<i>NewVelocity</i>	<b>InflIntrpOut (SharedInflowDefs)</b>			U,V,W velocities

#### STEPS

1. Initialize: *WindInf\_GetTI(:) = 0.0*
2. Set *Nt = (EndTime - StartTime) / delta\_time* ; number of time intervals
3. IF ( *Nt < 2* ) RETURN; 0 std. dev.
4. IF .NOT. ALLOCATED allocate *Velocity(3,Nt)*; in case of *ErrStat<>0* warn user and RETURN
5. Initialize: *SumAry(:) = 0.0*
6. DO\_LOOP *I=1,Nt* :

- a.  $Time = StartTime + (I-1)*delta\_time$ ; not used
  - b.  $NewVelocity = \underline{WindInf\_GetVelocity}(Time, InputPosition, ErrStat)$  (*InflowWind*); RETURN  
in case ErrStat<>0
  - c.  $Velocity(:,I) = NewVelocity \% Velocity(:)$
  - d.  $SumAry(:) = SumAry(:) + NewVelocity \% Velocity(:)$
7. END\_LOOP
8.  $MeanVel(:) = SumAry(:) / \text{REAL}(Nt, ReKi)$ ; this is the mean velocity
9. IF (  $\text{ABS}(MeanVel(1)) \leq \text{EPSILON}(MeanVel(1))$  ) THEN : too small a wind speed  
a. Warn user and RETURN
10. Reset  $SumAry(:) = 0.0$  to calc std. dev.
11. DO\_LOOP I=1,Nt
- a.  $SumAry(:) = SumAry(:) + (Velocity(:,I) - MeanVel(:))^{\star\star 2}$
12. END\_LOOP
13. Set Output:  $WindInf\_GetStedDev(:) = \text{SQRT}(SumAry(:) / (Nt-1)) / MeanVel(1)$

## OUTPUT

*WindInf\_GetTI(3): R(ReKi); Turbulence intensities for the components U, V, W of the velocity*

It returns the TIs of wind speed components U-V-W for the time interval and position requested, it makes use of specific subroutines to calculate the wind speed depending on the type of wind file.

## G6. WINDINF\_ADHACK\_DICHECK (ERRSTAT)

This function should be deleted ASAP. Its purpose is to reproduce results of AeroDyn 12.57; it performs a wind speed check for the dynamic inflow initialization; it returns MFFWS for the FF wind files; for all others, a sufficiently large number is used (> 8 m/s). Called by *AD\_Init* (*AeroDyn*).

### EXTERNAL (INVOKED) MODULES

See Parent Module *InflowWindMod.f90*

### INPUT

Name	Type/Module	INTENT	Description
<i>ErrStat</i>	I	OUT	Error Status Flag

Additional input is given through invoked module variables

### STEPS

1. Set *ErrStat*=0
2. If *WindType* (parent module *InflowWind*) = *HH\_Wind* or *UD\_Wind* or *FD\_Wind* Then:
  - a. *WindInf\_Adhack\_DIcheck* = 50
  - b. Else if *WindType* = *FF\_Wind*, Then *WindInf\_Adhack\_DIcheck* = *FF\_GetValue*('MEANFFWS', *ErrStat*) (*FFWind*)

### OUTPUT

*WindInf\_Adhack\_DIcheck*, R(ReKi). It returns a value depending on the type of wind file read to see whether or not to keep the dynamic inflow active.

## G7. WINDINF\_TERMINATE( ErrStat)

Subroutine that closes all open files and cleans up allocated variables for wind files. Resets the initialization flag so that we have to reinitialize before calling the routines again. Called by WindInf\_Init (*InflowWind*).

### EXTERNAL (INVOKED) MODULES

See Parent Module *InflowWind*[Mod.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>ErrStat</i>	I	OUT	Error Status Flag

Additional input is given through invoked module variables

### LOCAL VARIABLES

N/A

### STEPS

1. Close(*UnWind*) (*InflowWind*)
2. Based on *WindType*
  - a. *HH\_Wind*: Call HH\_Terminate(*ErrStat*) (*HHWind*)
  - b. *FF\_Wind*: Call FF\_Terminate(*ErrStat*) (*FFWind*)
  - c. *UD\_Wind*: Call UsrWnd\_Terminate(*ErrStat*) (*HHWind*)
  - d. *FD\_Wind*: Call FD\_Terminate(*ErrStat*) (*HHWind*)
3. Call CT\_Terminate(*ErrStat*) (*CTWind*)
4. Reset *WindType*=0 (*InflowWind*) so that the initialization routine must be called again in case.
5. Reset *CT\_Flag*=.FALSE. (*InflowWind*)

### OUTPUT

It cleans up by deallocating variables by calling the right subroutines depending on the wind file type. It also resets *CT\_Flag* and *WindType* variables.

## G8. GETWINDTYPE(FILENAME, ERRSTAT)

Function that checks the file *FileName* to see what kind of wind file we are using. Used when the wind file type is unknown. Called by WindInf Init (*InflowWind*).

### EXTERNAL (INVOKED) MODULES

See Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>FileName</i>	C(*)	INOUT	Filename for wind file
<i>ErrStat</i>	I	OUT	>0 if error encountered

Additional input is given through invoked module variables in parent module

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>IND</i>	I			Auxiliary index for string character
<i>Exists</i>	L			Auxiliary variable to check on summary file existence
<i>FileNameEnd</i>	C(3)			Auxiliary string for filename extention
<i>WndFilNam</i>	C(8)			Wind File name

#### STEPS

1. Initialize *ErrStat*=0
2. Initialize *WndFilNam*=*FileName* and convert to upper cases
3. If *WndFilName* =='USERWIND' then write to screen that user-defined wind file has been detected and set *GetWindType*= *UD\_Wind* (*SharedInflowDefs*) and RETURN
4. Get the file extension *FileNameEnd* from *FileName*
5. If no extension add extension “.wnd” and assume *GetWindType*= *FF\_Wind* (*SharedInflowDefs*); else:
  - a. If the extension is “.wnd” and the string ‘sum’ is within the *FileName*, then set *GetWindType*= *FF\_Wind* [binary FF wind file] else *GetWindType*= *HH\_Wind* (*SharedInflowDefs*) [formatted wind file]
  - b. If the extension is “.bts” then set *GetWindType*= *FF\_Wind*; [binary FF wind file]
  - c. If the extension is “.ctp” then set *GetWindType*= *CTP\_Wind* (*SharedInflowDefs*); [coherent turbulence wind file]
  - d. If the extension is “.fdp” then set *GetWindType*= *FD\_Wind* (*SharedInflowDefs*); [4D wind file]
  - e. All other cases assume *GetWindType*= *HH\_Wind* [formatted wind file]

## OUTPUT

*GetWindType: I;* Depending on file extension it returns the wind file type as an integer (code specified in *SharedInflowDefs*).

## G9. WINDINF\_ADHACK\_DISKVEL(TIME, INPPOSITION, ERRSTAT)

This function should be deleted ASAP. Its purpose is to reproduce results of AeroDyn 12.57; when a consensus on the definition of "average velocity" is determined, this function will be removed. InpPosition(2) should be the rotor radius; InpPosition(3) should be hub height. Called by DiskVel (AeroSubs).

### EXTERNAL (INVOKED) MODULES

See Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>Time</i>	C(*)	IN	Current time of interest to get velocity at
<i>InpPosition(3)</i>	R(ReKi)	IN	Should be disk center height (hub-height)
<i>ErrStat</i>	I	OUT	If $\neq 0$ , error encountered

Additional input is given through invoked module variables

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>NewVelocity</i>	<i>InflIntrpOut</i> <i>(SharedInflowDefns)</i>			U, V, W velocities
<i>Position(3)</i>	R(ReKi)			Position Vector in FF0
<i>IX</i>	I			Counter
<i>IZ</i>	I			Counter

#### STEPS

1. Initialize *ErrStat*=0
2. Based on *WindType* (parent module *InflowWind*):
  - a. If *HH\_Wind* (*SharedInflowDefns*):
    - i. *Position*=[ 0.0, 0.0, *InpPosition(3)* ]
    - ii. *NewVelocity* = *HH\_Get\_ADHack\_WindSpeed*(*Time*, *Position*, *ErrStat*) (*HHWind*)
    - iii. *WindInf\_Adhack\_diskVel*(:) = *NewVelocity*%*Velocity*(:)
  - b. If *FF\_Wind* (*SharedInflowDefns*):
    - i. *WindInf\_Adhack\_diskVel*(1) = *FF\_GetValue*('MeanFFWS', *ErrStat*) (*FFWind*)
    - ii. *WindInf\_Adhack\_diskVel*(2:3) = 0.0
  - c. If *UD\_Wind* (*SharedInflowDefns*):
    - i. *WindInf\_Adhack\_diskVel*(1) = *UsrWnd\_GetValue*('MeanU', *ErrStat*) (*UsrWind*)
    - ii. *WindInf\_Adhack\_diskVel*(2) = *UsrWnd\_GetValue*('MeanV', *ErrStat*) (*UsrWind*)
    - iii. *WindInf\_Adhack\_diskVel*(3) = *UsrWnd\_GetValue*('MeanW', *ErrStat*) (*UsrWind*)

- d. If *FD\_Wind* (*SharedInflowDefns*):
  - i. Initialize:
    - 1. *Position*(1)=0.0
    - 2. *WindInf\_ADhack\_diskVel*(:) = 0.0
  - ii. Cycle through IY=-1,1,2 and IZ=-1,1,2
    - 1. *Position*(2)= *IY*\**FD\_GetValue*('RotDiam',*ErrStat*) (*FDWind*)
    - 2. *Position*(3)= *IZ*\**InpPosition*(2)+ *InpPosition*(3)
    - 3. *NewVelocity* = *WindInf\_GetVelocity*(*Time*, *Position*, *ErrStat*) (parent module *InflowWind*)
    - 4. *WindInf\_ADhack\_diskVel*(:) = *WindInf\_ADhack\_diskVel*(:) + *NewVelocity*%*Velocity*(:)
  - iii. Reduce output to ¼: *WindInf\_ADhack\_diskVel*(:) \*0.25, a sort of average of 4 point velocities
- e. All other cases generate a warning, *ErrStat*=1 and RETURN

## OUTPUT

*WindInf\_ADhack\_diskVel*(3), R(ReKi); Depending on wind file type, it returns a sort of mean wind speed along the FF0 inertial coordinate system.

## G10. WINDINF\_LINEARIZEPERTURBATION(LINPERTURBATIONS, ERRSTAT)

SubRoutine used in FAST's linearization scheme. Called by FAST (external to AeroDyn).

### EXTERNAL (INVOKED) MODULES

See Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>LinPerturbations</i> (7)	R(ReKi)	IN	Input to <i>HH_SetLinearizeDels</i> ( <i>HHWind</i> )
<i>ErrStat</i>	I	OUT	Error Status Flag

Additional input is given through parent module and invoked module variables

#### LOCAL VARIABLES

N/A

#### STEPS

1. Initialize *ErrStat*=0
2. Based on *WindType* (parent module *InflowWind*):
  - a. *HH\_Wind*: Call *HH\_SetLinearizeDels* (*LinPerturbations*, *ErrStat*) (*HHWind*)
  - b. *FF\_Wind*, *UD\_Wind*, *FD\_Wind*: warn user that linearization works only for HH files, set *ErrStat*=1
  - c. All other cases: warn user that there is an undefined wind file type, and set *ErrStat*=1

#### OUTPUT

*ErrStat* : I; usual error status flag.

It sets (via a call to *HH\_SetLinearizeDels* (*HHWind*) ) other variables belonging to the *HHWind* module, see that module.

## APPENDIX H. SHAREDINFLOWDEFS MODULE

### GENERAL ORGANIZATION

SharedInflowDefs.f90 contains the module *SharedInflowDefs*, which includes a series of new parameters, and one new type.

This module is used to define shared types and parameters used in module *InflowWind* [see InflowWindMod.f90].

### EXTERNAL (INVOKED) MODULES

- *NWTC\_Library* (for the *Precision* module).

### PUBLIC (LOCALLY DECLARED) DATA

- **InflIntrpOut**: PUBLIC NEW TYPE made up of 1 real array:

Name	Type	Value	Parameter/ Initialized	Description
<i>Velocity</i> (3)	Re(ReKi)			U, V, W

- New PUBLIC Parameters:

Name	Type	Value	Parameter/ Initialized	Description
<i>DEFAULT_Wind</i>	I	-1	PARAMETER	Undetermined wind type; calls internal routine to guess what type of file it is.
<i>HH_Wind</i>	I	1	PARAMETER	Hub-Height wind file
<i>FF_Wind</i>	I	2	PARAMETER	Binary full-field wind file
<i>UD_Wind</i>	I	3	PARAMETER	User-defined wind
<i>FD_Wind</i>	I	4	PARAMETER	4-dimensional wind (LES)
<i>CTP_Wind</i>	I	5	PARAMETER	Coherent turbulence wind field (superimpose KH billow on background wind)

## APPENDIX I. HHWIND MODULE

### GENERAL ORGANIZATION

HHWind.f90 contains the *HHWind* module, which includes all the data and procedures that define hub-height wind files.

### THEORY

The HH wind files could more accurately be called point wind files since the wind speed at any point is calculated by shear applied to the point where the wind is defined. It is basically uniform wind over the rotor disk. The entire file is read on initialization, then the columns that make up the wind file are interpolated to the time requested, and wind is calculated based on the location in space.

The wind file contains header information (rows that contain "!"'), followed by numeric data stored in 8 columns:

1. Time [s]
2. Horizontal wind speed (V) [m/s]
3. Wind direction (Delta) [deg]
4. Vertical wind speed (VZ) [m/s]
5. Horizontal linear shear (HLinShr) [-]
6. Vertical power-law shear (VShr) [-]
7. Vertical linear shear (VLinShr) [-] (if this is activated then put VShr=0)
8. Gust (horizontal) velocity (VGust) [m/s]

The horizontal wind speed at (X, Y, Z) is then calculated using the interpolated columns by:

$$\begin{aligned}
 Vh &= V * (Z/\text{RefHt})^{**} VShr && \text{(i.e., power-law wind shear)} \\
 &+ V * HLinShr/\text{RefWid} * (Y * \text{COS}(\text{Delta}) + X * \text{SIN}(\text{Delta})) && \text{(i.e., horizontal linear shear)} \\
 &+ V * VLInShr/\text{RefWid} * (Z - \text{RefHt}) && \text{(i.e., vertical linear shear)} \\
 &+ VGust && \text{(i.e., gust speed)}
 \end{aligned}$$

### EXTERNAL MODULES

- *NWTC\_Library* [see NWTC\_Library.f90]
- *SharedInflowDefns* [see SharedInflowDefs.f90]

### PRIVATE (LOCALLY DECLARED) DATA

Name	Type	Value	Parameter/ Initialized	Description
<i>Tdata</i> (:)	R(ReKi)			Time array from the HH wind file
<i>Delta</i> (:)	R(ReKi)			HH Wind direction (angle)
<i>V</i> (:)	R(ReKi)			HH horizontal wind speed
<i>VZ</i> (:)	R(ReKi)			Wind, including tower shadow, along the Z axis

<i>HSHR</i> (:)	R(ReKi)	HH Horizontal linear shear
<i>VSHR</i> (:)	R(ReKi)	HH vertical shear exponent
<i>VLINSHR</i> (:)	R(ReKi)	HH vertical linear shear
<i>VGUST</i> (:)	R(ReKi)	HH wind gust
<i>LinearizeDels</i> (7)	R(ReKi)	The delta values for linearization -- perhaps at some point, this could be T/F and we determine the deltas by sqrt(eps) or something similar
<i>RefHt</i>	R(ReKi)	Reference height; was HH (hub height); used to center the wind
<i>RefWid</i>	R(ReKi)	Reference width; used to be 2*R (=rotor diameter); used to scale the linear shear
<i>NumDataLines</i>	I	Number of data lines in the file
<i>TimeIndx</i>	I	0      Initialized, SAVE      An index into the <i>Tdata</i> array (to allow us faster searching, starting search from previous one)
<i>Linearize</i>	L	.FALSE.      Initialized, SAVE      If this is TRUE, we are linearizing

### PUBLIC (LOCALLY DECLARED) DATA

- **HH\_Info:** NEW PUBLIC TYPE made up of 2 real fields:

Name	Type	Value	Parameter/ Initialized	Description
<i>ReferenceHeight</i>	R(ReKi)			Reference height (HH)
<i>Width</i>	R(ReKi)			Reference Width

- New PUBLIC Routines and Functions

Name	Description
<u><i>HH_Init</i></u>	Initialization of the module and read input
<u><i>HH_SetLinearizeDels</i></u>	Perturbation (see FAST)
<u><i>HH_GetWindSpeed</i></u>	Wind Velocity at time and position of interest
<u><i>HH_Get_Adhack_WindSpeed</i></u>	To get a normal to rotor, spatially averaged, wind speed
<u><i>HH_Terminate</i></u>	Terminate (Deallocate) module

## 11. HHINIT (UNWIND, WINDFILE, WINDINFO, ERRSTAT)

Subroutine that reads the HH file and stores the data in an array to use later. It requires an initial reference height (hub height) and width (rotor diameter), both in meters, which are used to define the volume where wind velocities will be calculated. This information is necessary because of the way the shears are defined. Called by *WindInf Init (InflowWind)*.

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for reading wind files
<i>WindFile</i>	C(*)	IN	Name of the HH file (text file)
<i>WindInfo</i>	<b>HH_Info (HHWind)</b>	IN	Additional information needed to initialize this wind type
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *TimeIndx (SharedInflowDefns)*)

#### LOCAL VARIABLES

Name	Type/ Module	Value	Parameter/ Initialized	Description
<i>NumCols</i>	I	8	PARAMETER	Number of columns in the HH file
<i>TmpData(NumCols)R(ReKi)</i>				Temp. variable for reading all columns from a line
<i>DelDiff</i>	R(ReKi)			Temp. variable for storing the direction difference
<i>I</i>	I			Counter
<i>NumComments</i>	I			Number of comment lines in the HH file
<i>ILine</i>	I			Counter
<i>MaxTries</i>	I	100	PARAMETER	Max no. of cycles to check no jumps greater than 180° exist
<i>Line</i>	C(1024)			Temp. variable for reading whole line from file

#### STEPS

1. IF *TimeIndx* (parent module *HHWind*) $\neq 0$ : module already initialized: THEN:

- a. warn user and RETURN with *ErrStat*=1
2. ELSE:
  - a. *ErrStat*=0 and Call **NWTC\_Init (NWTC\_Library)**
  - b. Initialize *LinearizeDels*(:) =0.0 and *Linearize* =.FALSE. (***HHWind***)
3. Open the file: Call **OpenFInpFile (UnWind, TRIM(WindFile), ErrStat) (NWTC\_IO** within ***NWTC\_Library***)
4. Read all lines containing the comment character “ ! ” with a simple DO WHILE, and put the number of comment lines into *NumComments*
5. Read all lines containing data : READ(*UnWind*,\*,IOSTAT=*ErrStat*) ( *TmpData(I)*, *I*=1,*NumCols* ) and put number of those lines into *NumDataLines* (***HHWind***)
6. IF *NumDataLines* < 1 THEN warn user and RETURN ELSE notify user of now many data lines are present
7. Allocate arrays for the HH data: IF NOT ALLOCATED ALLOCATE THE FOLLOWING:
  - a. *Tdata(NumDataLines)*, *V(NumDataLines)*, *Delta(NumDataLines)*, *VZ(NumDataLines)*, *HSHR(NumDataLines)*, *VSHR(NumDataLines)*, *VLinShr(NumDataLines)*, *VGUST(NumDataLines)* (***HHWind***)
8. REWIND (*UnWind*) unit file and read comment lines using **ReadCom (NWTC\_IO** within ***NWTC\_Library***)
9. DO\_LOOP *I*=1,*NumDataLines*: Now finally read arrays:
  - a. Call **ReadAry( UnWind, TRIM(WindFile), TmpData(1:*NumCols*), NumCols, 'TmpData', 'Data from HH line '//TRIM(Int2LStr(*NumComments*+*I*)), ErrStat )** (***NWTC\_IO*** within ***NWTC\_Library***)
  - b. Assign *Tdata(I)* through *VGUST(I)* appropriately as *TmpData*(1 through 8), also transforming *Delta* into radians
10. END\_LOOP
11. DO\_LOOP *I*=2,*NumDataLines*: Make sure we do not have 180° jumps in wind directions
  - a. Initialize *ILine*=1
  - b. DO\_LOOP *ILine* < *MaxTries*
    - i. *DelDiff* = ( *Delta(I)* - *Delta(I-1)* ) (jump between 2 consecutive wind dirs.)
    - ii. IF *DelDiff* < *Pi* (***NWTC\_Num***) THEN EXIT Loop
    - iii. ELSE :
      1. *Delta(I)* = *Delta(I)* - SIGN( *TwoPi* (***NWTC\_Num***), *DelDiff* )
      2. *ILine* = *ILine* + 1
  - c. END\_LOOP This loop makes sure we have an unwrapped sequence of *Delta* but it does not put it within [0, 2*Pi*]
12. Check IF *ILine* > *MaxTries* in which case THEN warn user and set *ErrStat* =1
13. Close (UnWind) File unit
14. IF *Tdata*(1) > 0.0 (start time > 0.0) THEN warn user of possible problems with time interpolations.
15. IF *NumDataLines* ==1 THEN warn user that a Steady State Wind is assumed
16. Set *TimeIndx*=1 (It indicates the module has been initialized)

17. Set  $RefHt$  (**HHWind**) =  $WindInfo\%ReferenceHeight$
18. Set  $RefWid$  (**HHWind**) =  $WindInfo\%Width$

## OUTPUT

$ErrStat : I$ ; usual error status flag.

It reads the HH type file and sets variables within **HHWind**:  $NumDatalines$ ,  $Tdata(NumDataLines)$ ,  $V(NumDataLines)$ ,  $Delta(NumDataLines)$ ,  $VZ(NumDataLines)$ ,  $HSHR(NumDataLines)$ ,  $VSHR(NumDataLines)$ ,  $VLinShr(NumDataLines)$ ,  $VGUST(NumDataLines)$

It also sets certain variables needed within **HHWind**:  $TimeIndx$ ,  $RefHT$ ,  $RefWid$ .

## I2.HH\_SETLINEARIZEDDELS (PERTURBATIONS, ERRSTAT)

Subroutine that sets the perturbation values to the linearization scheme. Called by *WindInf LinearizePerturbation (InflowWind)*.

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>Perturbations(7)</i>	R(ReKi)	IN	Perturbations for each of the 7 input parameters
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered (wrong comment in the code)

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Linearize (HHWind)*)

### LOCAL VARIABLES

N/A

### STEPS

1. IF *TimeIdx (HHWind)* ==0 (module not initialized) THEN:
  - a. warn user on the screen
  - b. Set *ErrStat*=1
  - c. RETURN
2. ELSE Initialize *ErrStat*=0
  - a. Set *Linearize* = .TRUE. (*HHWind*)
  - b. Set *LinearizeDels(:) (HHWind)* = *Perturbations (:)*

### OUTPUT

*ErrStat*: I; the usual error flag.

It also sets *Linearize*: L; linearization flag (*HHWind*) and *LinearizeDels(:)* : R(ReKi) (*HHWind*) perturbation values.

### 13. HH\_GETWINDSPEED (TIME, INPUTPOSITION, ERRSTAT)

This function linearly interpolates the columns in the HH input file to get the values for the requested time, then uses the interpolated values to calculate the wind speed at a point. Called by WindInf\_getVelocity (InflowWind).

#### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>Time</i>	R(ReKi)	IN	Time from the start of the simulation
<i>InputPosition (3)</i>	R(ReKi)	IN	Position (X, Y, Z)
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *TimeIndx (HHWind)*)

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>CosDelta</i>	R(ReKi)			Cosine of <i>Delta_tmp</i>
<i>Delta_tmp</i>	R(ReKi)			Interpolated <i>Delta</i> at input TIME
<i>HShr_tmp</i>	R(ReKi)			Interpolated <i>HShr</i> at input TIME
<i>P</i>	R(ReKi)			Temporary storage for slope (in time) used in linear interpolation
<i>SinDelta</i>	R(ReKi)			Sine of <i>Delta_tmp</i>
<i>V_tmp</i>	R(ReKi)			Interpolated <i>V</i> at input TIME
<i>VGust_tmp</i>	R(ReKi)			Interpolated <i>VGUST</i> at input TIME
<i>VLinShr_tmp</i>	R(ReKi)			Interpolated <i>VLinShr</i> at input TIME
<i>VShr_tmp</i>	R(ReKi)			Interpolated <i>VSHR</i> at input TIME
<i>VZ_tmp</i>	R(ReKi)			Interpolated <i>VZ</i> at input TIME
<i>V1</i>	R(ReKi)			Temporary storage for horizontal velocity

#### STEPS

1. Make sure that HHWind module is initialized:
  - a. IF *TimeIndx (HHWind)* ==0 (not initialized) THEN:

- i. warn user on the screen
  - ii. Set *ErrStat*=1
  - iii. RETURN
  - b. ELSE initialize *ErrStat*=0
2. Linearly Interpolate in Time:
3. IF *Linearize (HHWind)* THEN: get perturbed wind speed
- a. *TimeIndx*=1
  - b. *V\_tmp*= *V*(1) + *LinearizeDels*(1) (*HHWind*) ; note *LinearizeDels* are initialized in *HHInit*, but can also be modified by *HH\_SetLinearizedDels* (*HHWind*) called by *WindInf\_LinearizePerturbation* (*InflowWind*)
  - c. *Delta\_tmp* = *Delta* (1) + *LinearizeDels*(2) (*HHWind*)
  - d. *VZ\_tmp* = *VZ* (1) + *LinearizeDels*(3) (*HHWind*)
  - e. *HShr\_tmp* = *HShr* (1) + *LinearizeDels*(4) (*HHWind*)
  - f. *VShr\_tmp* = *VShr* (1) + *LinearizeDels*(5) (*HHWind*)
  - g. *VLinShr\_tmp* = *VLinShr*(1) + *LinearizeDels*(6) (*HHWind*)
  - h. *VGust\_tmp* = *VGust* (1) + *LinearizeDels*(7) (*HHWind*)
4. ELSEIF *Time*<= *Tdata*(1) .OR. *Time*>= *Tdata*(*NumDataLines*) :check that we are within Time Boundaries and in case assign the variables accordingly as start/end of arrays:
- a. *TimeIndx* = [1/*NumDataLines*-1]
  - b. *V\_tmp* = *V* ([1/*NumDataLines*])
  - c. *Delta\_tmp* = *Delta* ([1/*NumDataLines*])
  - d. *VZ\_tmp* = *VZ* ([1/*NumDataLines*])
  - e. *HSHR\_tmp* = *HSHR* ([1/*NumDataLines*])
  - f. *VSHR\_tmp* = *VSHR* ([1/*NumDataLines*])
  - g. *VLinSHR\_tmp* = *VLinSHR* ([1/*NumDataLines*])
  - h. *VGust\_tmp* = *VGUST* ([1/*NumDataLines*])
5. ELSE: do simple interpolation: (should be called with external *lininterp* routine)
- a. Set *TimeIndx* = MAX( MIN( *TimeIndx*, *NumDataLines*-1 ), 1 )
  - b. DO\_LOOP
    - i. IF *Time* < *Tdata*(*TimeIndx*) THEN Set *TimeIndx* = *TimeIndx* - 1
    - ii. ELSEIF *Time* >= *Tdata*(*TimeIndx*) THEN Set *TimeIndx* = *TimeIndx* + 1
    - iii. ELSE do the interpolation:
      - 1. Set *P* = (*Time* - *Tdata*(*TimeIndx*)) / ( *Tdata*(*TimeIndx*+1) - *Tdata*(*TimeIndx*) )
      - 2. For all above (b-h points) vectors (*V\_tmp* through *VGust\_tmp*) it is  
 $\#_tmp = (\#(TimeIndx+1) - \#(TimeIndx)) * P + \#(TimeIndx)$  (actual interpolation)
      - 3. EXIT LOOP
    - c. END\_LOOP
  - 6. Set *CosDelta*= COS(*Delta\_tmp*) and *SinDelta*= SIN (*Delta\_tmp*)
  - 7. Now calculate wind velocities at the requested time:

- a.  $V1 = V_{tmp} * ( ( InputPosition(3)/RefHt ) ** VShr_{tmp} + (HShr_{tmp} * ( InputPosition(2) * CosDelta + InputPosition(1) * SinDelta ) + VLInShr_{tmp} * ( InputPosition(3)-RefHt ) /RefWid ) + VGUST_{tmp}$
8. Set OUTPUT:
- $HH\_GetWindSpeed\%Velocity(1) = V1 * CosDelta$
  - $HH\_GetWindSpeed\%Velocity(2) = -V1 * SinDelta$
  - $HH\_GetWindSpeed\%Velocity(3) = VZ_{tmp}$

## OUTPUT

*ErrStat: I;* the usual error flag.

*HH\_GetWindSpeed:* TYPE(**InflIntrpOut**) (*SharedInflowDefns*); the wind velocity at the point of interest following the HH file type.

It also updates *TimeIndx: I, (HHWind)*.

## I4. HH\_GET\_ADHACK\_WINDSPEED (TIME, INPUTPOSITION, ERRSTAT)

This Function is used to linearly interpolate the columns in the HH input file to get the values for the requested time at the reference point (HH). THIS FUNCTION SHOULD BE REMOVED!!!! (used for DISK VEL ONLY). Called by WindInf ADhack DiskVel (*InflowWind*).

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>Time</i>	R(ReKi)	IN	Time from the start of the simulation
<i>InputPosition(3)</i>	R(ReKi)	IN	Input position (X,Y,Z): NOT USED AT ALL
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *TimeIndx (HHWind)*)

### LOCAL VARIABLES

Name	Type/Module	Parameter/ Value Initialized	Description
<i>Delta_Tmp</i>	R(ReKi)		Interpolated Delta at input TIME
<i>P</i>	R(ReKi)		Temporary storage for slope (in time) used in linear interpolation
<i>V_tmp</i>	R(ReKi)		Interpolated V at input TIME
<i>VZ_tmp</i>	R(ReKi)		Interpolated VZ at input TIME

### STEPS

1. Make sure that HHWind module is initialized:
  - a. IF *TimeIndx (HHWind)* ==0 (not initialized) THEN:
    - i. warn user on the screen
    - ii. Set *ErrStat*=1
    - iii. RETURN
  - b. ELSE initialize *ErrStat*=0
2. IF *Time* <= *Tdata(1)* .OR. *Time* >= *Tdata(NumDataLines)* :check that we are within Time Boundaries and in case assign the variables accordingly as start/end of arrays:
  - a. *TimeIndx* = [1/*NumDataLine*-1]
  - b. *V\_tmp* = *V* ([1/*NumDataLines*])
  - c. *Delta\_tmp* = *Delta* ([1/*NumDataLines*])
  - d. *VZ\_tmp* = *VZ* ([1/*NumDataLines*])
3. ELSE: do simple interpolation: (should be called with external lininterp routine)

- a. Set  $TimeIndx = \text{MAX}(\text{MIN}(TimeIndx, NumDataLines-1), 1)$
- b. DO\_LOOP
  - i. IF  $Time < Tdata(TimeIndx)$  THEN Set  $TimeIndx = TimeIndx - 1$
  - ii. ELSEIF  $Time \geq Tdata(TimeIndx)$  THEN Set  $TimeIndx = TimeIndx + 1$
  - iii. ELSE do the interpolation:
    - 1. Set  $P = (Time - Tdata(TimeIndx)) / (Tdata(TimeIndx+1) - Tdata(TimeIndx))$
    - 2. For all vectors  $(V_{tmp}, Delta_{tmp}, VZ_{tmp})$  it is  
 $\#_{tmp} = (\#(TimeIndx+1) - \#(TimeIndx)) * P + \#(TimeIndx)$  (actual interpolation)
    - 3. EXIT LOOP
  - c. END\_LOOP
- 4. Set OUTPUT:
  - a.  $HH\_Get\_ADHack\_WindSpeed\%Velocity(1) = V_{tmp} * \cos(Delta_{tmp})$
  - b.  $HH\_Get\_ADHack\_WindSpeed\%Velocity(2) = -V_{tmp} * \sin(Delta_{tmp})$
  - c.  $HH\_Get\_ADHack\_WindSpeed\%Velocity(3) = VZ_{tmp}$

## OUTPUT

*ErrStat: I;* the usual error flag.

*HH\_Get\_ADHack\_WindSpeed:* TYPE(**InflIntrpOut**) (*SharedInflowDefns*); the wind velocity at the HH point of reference following the HH file type, and along inertial reference frame

It also updates *TimeIndx: I, (HHWind)*.

## I5. HH\_TERMINATE (ERRSTAT)

Subroutine to close files, deallocate memory, and un-set the initialization flag. Called by WindInf\_Terminate (InflowWind).

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Tdata (HHWind)*)

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>SumErrs</i>	I			Loop counter

#### STEPS

1. Set *SumErrs*=0
2. DEALLOCATE if allocated :
  - a. *Tdata, DELTA, V, VZ, HSHR, VSHR, VGUST, VLINSHR (HHWind)*
  - b. In the process augment *SumErrs* with the ABS(*ErrStat*) returned by DEALLOCATE
3. Set *ErrStat*= *SumErrs*
4. Set *TimeIndx*=0; initialization flag: if 1=initialized

#### OUTPUT

*ErrStat*: I; the usual error flag

It deallocates memory.

## APPENDIX J. FFWIND MODULE

### GENERAL ORGANIZATION

This module uses full-field binary wind files to determine the wind inflow. This module assumes that the origin, (0,0,0), is located at the tower centerline at ground level, and that all units are specified in the metric system (using meters and seconds). Data is shifted by half the grid width to account for turbine yaw (so that data in the X direction actually starts at -1\*FFYHWid meters)

#### External Modules

- *NWTC\_Library* [see NWTC\_Library.f90]
- *SharedInflowDefns* [see SharedInflowDefs.f90]

#### PRIVATE (LOCALLY DECLARED) DATA

Name	Type	Value	Parameter/ Initialized	Description
<i>FFData(:,:,:,:)</i>	R(ReKi)			Array of FF data ( <i>NZGrids,NYGrids,NFFComp, NFFSteps</i> )
<i>FFtower (:,:,:,)</i>	R(ReKi)			Array of data along the tower, ( <i>NFFComp, NTgrids, NFFSteps</i> )
<i>FFDTime</i>	R(ReKi)			Delta time
<i>FFRate</i>	R(ReKi)			Data rate in Hz (1/ <i>FFDTime</i> )
<i>FFYHWid</i>	R(ReKi)			Half the grid width
<i>FFZHWid</i>	R(ReKi)			Half the grid height
<i>RefHt</i>	R(ReKi)			Reference (hub) height of the grid in meters
<i>GridBase</i>	R(ReKi)			Height of the bottom of the grid in meters
<i>InvFFYD</i>	R(ReKi)			1/ delta-y
<i>InvFFZD</i>	R(ReKi)			1/ delta-z
<i>InvMFFWS</i>	R(ReKi)			Reciprocal of the mean wind speed ( <i>MeanFFWS</i> )
<i>MeanFFWS</i>	R(ReKi)			Mean wind speed (as defined in the FF file), not necessarily the mean of the portion of the wind used.
<i>NFFComp</i>	I			Number of wind components
<i>NFFSteps</i>	I			Number of time steps in the FF array
<i>NYGrids</i>	I			Number of points in the lateral (y) grid-direction

<i>NZGrids</i>	I	Number of points in the vertical (z) grid-direction
<i>NTGrids</i>	I	Number of points in the vertical (z) direction on the tower (below the grids)
<i>Initialized</i>	L .FALSE. SAVE	Flag that determines if the module has been initialized

- New PRIVATE Routines and Functions

<u>Read_TurbSim_FF</u>	It reads TurbSim data file
<u>FF_Interp</u>	It interpolates wind data at point and time requested
<u>Read_FF_Tower</u>	It reads Tower wind data file associated with BLADED style file
<u>ReadBladed_FF_Header[0/1]</u>	They read old or new BLADED style header info from BLADED style file
<u>ReadBladed_Grids</u>	It reads wind data from BLADED style file
<u>Read_Summary_FF</u>	It reads summary file associated with wind data from BLADED style file

## PUBLIC (LOCALLY DECLARED) DATA

- New PUBLIC Routines and Functions

Name	Description
<u>FF_Init</u>	Initialize module and reads in the wind data files
<u>FF_GetWindSpeed</u>	Interpolates the wind velocity components and returns them for the points and time of interest form the wind data file
<u>FF_GetValue</u>	INTERFACE MODULE TO PROCEDURE <u>FF_GetRValue</u> Return values of requested scalar variables
<u>FF_Terminate</u>	Terminates and deallocates module

## J1. FFINIT (UNWIND, BINFILE, ERRSTAT)

Subroutine that reads the full-field turbulence data called at the beginning of a simulation. Called by WindInf Init (InflowWind).

### EXTERNAL (INVOKED) MODULES

From Parent Module *FFWind*[.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for reading wind files
<i>BinFile</i>	C(*)	IN	Name of the binary FF wind file
<i>ErrStat</i>	I	OUT	If $<>0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Initialized (FFWind)*)

#### LOCAL VARIABLES

Name	Type	Value	Parameter/ Initialized	Description
<i>TI(3)</i>	R(ReKi)			Turbulence intensities of the wind components as defined in the FF file, not necessarily the actual TI
<i>BinTI(3)</i>	R(ReKi)			Turbulence intensities of the wind components as defined in the FF binary file, not necessarily the actual TI
<i>UBAR</i>	R(ReKi)			Mean wind speed
<i>ZCenter</i>	R(ReKi)			Output of <u>Read_Summary_FF (FFWind)</u>
<i>Dum_Int2</i>	I(B2Ki)			Dummy variable
<i>DumInt</i>	I			Dummy variable but simple integer
<i>I</i>	I			Counter
<i>CWise</i>	L			If clockwise rotation
<i>Exists</i>	L			Variable set when file searched
<i>SumFile</i>	C(1028)			Length is LEN(BinFile) + the 4-character extension.
<i>TwrFile</i>	C(1028)			Length is LEN(BinFile) + the 4-character extension.

#### STEPS

1. IF *Initialized* (parent module *FFWind*): module already initialized: THEN:

- a. warn user and RETURN with *ErrStat*=1

2. ELSE *ErrStat*=0 and Call *NWTC\_Init* (*NWTC\_Library*)
3. Read one variable out of the input binary file:
  - a. Call *OpenBInpFile*( *UnWind*, TRIM(*BinFile*), *ErrStat* ) (*NWTC\_IO*), RETURN in case *ErrStat* is returned  $\neq 0$
  - b. READ(*UnWind*) *Dum\_Int2*
  - c. CLOSE(*UnWind*)
  - d. Set *DumInt* = *Dum\_Int2* so that we have a simple integer to use later in the check
4. Based on *DumInt* :
  - a. 7: TurbSim binary format: Call *Read\_TurbSim\_FF*(*UnWind*, TRIM(*BinFile*), *ErrStat*) (*FFWind*)
  - b. -1, -2, -3, OR -99: BLADED-Style binary format:
    - i. Get the file name root: Call *GetRoot*(*BinFile*, *SumFile*) (*NWTC\_IO* within *NWTC\_Library*)
    - ii. From the file root name create *TwrFile* = TRIM(*SumFile*)//'.tvr' and *SumFile* = TRIM(*SumFile*)//'.tvr'
    - iii. Read the summary file: Call *Read\_Summary\_FF* (*UnWind*, TRIM(*SumFile*), *CWise*, *ZCenter*, *TI*, *ErrStat*) (*FFWind*) and RETURN in case *ErrStat* $\neq 0$
    - iv. Set *UBar*=*MeanFFWS* (*FFWind*) store the mean wind speed
    - v. Now open the binary file again Call *OpenBInpFile*( *UnWind*, TRIM(*BinFile*), *ErrStat* ) (*NWTC\_IO*), RETURN in case *ErrStat* is returned  $\neq 0$
    - vi. IF ( *Dum\_Int2* == -99 ) THEN: this means NEW-style for BLADED format
      1. Call *Read\_Bladed\_FF\_Header1* (*UnWind*, *BinTi*, *ErrStat*) (*FFWind*) to read header
      2. DO\_LOOP *I*=1,*NFFComp* (*FFWind*) : if *BinTi* is present in the binary file use that rather than numbers from summary file
        - a. IF ( *BinTi*(*I*) > 0 ) *TI*(*I*) = *BinTi*(*I*)
    - vii. ELSE: this means OLD style for BLADED format: read the headers
      1. Call *Read\_Bladed\_FF\_Header0* (*UnWind*, *ErrStat*) (*FFWind*) and RETURN in case *ErrStat* is returned  $\neq 0$
    - viii. Check that *UBar* and *MeanFFWS* match if not RETURN with *ErrStat*=1
    - ix. Set *GridBase* (*FFWind*) = *ZCenter* – *FFZHWid* (*FFWind*) i.e. the bottom of the grid height in meters
    - x. Call *Read\_Bladed\_Grids*( *UnWind*, *CWise*, *TI*, *ErrStat* ) (*FFWind*): keep reading the file after headers have been read
    - xi. Close the file unit *UnWind*; RETURN in case *ErrStat* is returned  $\neq 0$  from previous call
    - xii. Read the Tower File if it exists:
      1. INQUIRE ( FILE=TRIM(*TwrFile*), EXIST=*Exists* )
      2. IF *Exists* THEN Call *Read\_FF\_Tower*( *UnWind*, TRIM(*TwrFile*), *ErrStat* ) (*FFWind*)

3. ELSE set  $NTgrids = 0$  (*FFWind*)
- c. Any other vase warn the user of unrecognized wind file type, RETURN with  $ErrStat=1$
5. Set *Initialized* = .TRUE. ; Initialization Flag ;

## OUTPUT

*Initialized*: (*FFWind*) Initialization flag

*ErrStat* : I; the usual error status flag.

It also sets certain variables needed within *FFWind* from called subroutines (e.g.: *MeanFFWS*, *FFZHWid*, *NTgrids*, etc.).

## J2. READBLADED\_FF\_HEADER0 (UNWIND, ERRSTAT)

Subroutine to read the binary headers from the turbulence files of the OLD BLADED STYLE. Note that because of the normalization, neither NZGrids or NYGrids are larger than 32 points.. Called by *FFInit* (*FFWind*).

### EXTERNAL (INVOKED) MODULES

From Parent Module *FFWind*[.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for reading wind files
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

### LOCAL VARIABLES

Name	Type	Value	Parameter/ Initialized	Description
<i>FFXDelt</i>	R(ReKi)			Temporary variable to construct others
<i>FFYDelt</i>	R(ReKi)			Temporary variable to construct others
<i>FFZDelt</i>	R(ReKi)			Temporary variable to construct others
<i>Dum_Int2</i>	I(B2Ki)		Dummy variable to read from file various variables	
<i>I</i>	I		Counter	

### STEPS

1. Read in sequence (multiple Read *Dum\_Int2* from *UnWind*) the file and get values for various variables declared in *FFWind* (various *ErrStat*  $\neq 0$  are preformed and in case user is warned and RETURN):
  - a. Set *NFFComp* = - *Dum\_Int2* ; -NFFC (file ID)
  - b. Set *FFZDelt* = 0.001\**Dum\_Int2* ; delta z (mm)
  - c. Set *InvFFZD* = 1.0/*FFZDelt* ;
  - d. Set *FFYDelt* = 0.001\**Dum\_Int2* ; delta y (mm)
  - e. Set *InvFFYD* = 1.0/*FFYDelt* ;
  - f. Set *FFXDelt* = 0.001\**Dum\_Int2* ; delta x (mm)
  - g. Set *NFFSteps* = 2\**Dum\_Int2* ; number of time steps
  - h. Set *MeanFFWS* = 0.1\**Dum\_Int2* ; mean full-field wind speed
  - i. Set *InvMFFWS* = 1.0/*MeanFFWS*
  - j. Set *FFDTime* = *FFXDelt*/*MeanFFWS*
  - k. Set *FFRate* = 1.0/*FFDTime*
  - l. Skip through the next 5 variables (not used)
  - m. Set *NZGrids* = *Dum\_Int2*/1000

- n. Set  $FFZHWid = 0.5 * FFZDelt * (NZGrids - 1)$  ; half the vertical size of the grid
- o. Set  $NYGrids = Dum\_Int2 / 1000$
- p. Set  $FFYHWid = 0.5 * FFYDelt * (NYGrids - 1)$
- q. IF ( $NFFComp == 3$ ) THEN Skip through the next 6 variables (not used)

## OUTPUT

It reads the OLD BLADED STYLE file and sets several variables belonging to the module ***FFWind***: *NFFComp*, *InvFFZD*, *InvFFYD*, *NFFSteps*, *MeanFFWS*, *InvMFFWS*, *FFDTime*, *FFRate*, *NZGrids*, *FFZHWid*, *NYGrids*, *FFYHWid*.

*ErrStat : I*; the usual error status flag.

### J3. READBLADED\_FF\_HEADER1 (UNWIND, TI, ERRSTAT)

Subroutine to read the binary headers from the turbulence files of the NEW BLADED STYLE. Called by ***FFInit (FFWind)***.

#### EXTERNAL (INVOKED) MODULES

From Parent Module ***FFWind.f90***

#### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for reading wind files
<i>ErrStat</i>	I	OUT	If $<>0$ , errors encountered
<i>TI(3)</i>	R(ReKi)	OUT	Turbulence Intensity components

#### LOCAL VARIABLES

Name	Type	Value	Parameter/ Initialized	Description
<i>FFXDelta</i>	R(ReKi)			Temporary variable to construct others
<i>FFYDelta</i>	R(ReKi)			Temporary variable to construct others
<i>FFZDelta</i>	R(ReKi)			Temporary variable to construct others
<i>Dum_Int2</i>	I(B2Ki)			Dummy variable to read from file various variables
<i>Dum_Int4</i>	I(B4Ki)			Dummy variable to read from file various variables
<i>Dum_Real4</i>	R(SiKi)			Dummy variable to read from file various variables
<i>I</i>	I			Counter
<i>TurbType</i>	I			Turbulence Spectrum flag

#### STEPS

1. Initialize  $TI(:)=-1$  ; in fact TI may not be included in the file
2. Read *Dum\_Int2* from *UnWind* file (IF *ErrStat*  $<>0$  user is warned and RETURN):
3. Set *TurbType* = *Dum\_Int2* ; turbulence type
4. Based on *TurbType*:
  - a. 1 or 2: Set *NFFComp (FFWind)*=1 (1-comp. Von-Karman or Kaimal)
  - b. 3 or 5: Set *NFFComp*=3 (3-comp. Von-Karman or IEC-2 Kaimal)
  - c. 4: Improved Von Karman
    - i. Read *Dum\_Int4* from *UnWind* file (IF *ErrStat*  $<>0$  user is warned and RETURN):
    - ii. Set *NFFComp* = *Dum\_Int4*
    - iii. Skip the next three variables (not used)

- iv. Read  $TI$ : DO\_LOOP I=1,3: Read  $Dum\_Real4$  from  $UnWind$  file (IF  $ErrStat < 0$  user is warned and RETURN): Set  $TI(I) = Dum\_Real4$ ; note this overwrites what was read in the summary file for  $TI$
  - d. 7 or 8: General Kaimal (7) or Mann model (8)
    - i. Skip one variable
    - ii. Read  $Dum\_Int4$  from  $UnWind$  file (IF  $ErrStat < 0$  user is warned and RETURN):
    - iii.  $NFFComp = Dum\_Int4$
  - e. All other cases : warn user of unrecognized full field turbulence file type
5. Read in sequence (multiple Read  $Dum\_Real4$  and  $Dum\_Int4$  from  $UnWind$ ) the file and get values for various variables declared in ***FFWind*** (various  $ErrStat > 0$  are preformed and in case user is warned and RETURN):
- a. Set  $FFZDelt = Dum\_Real4$  ; delta-z (m)
  - b. Set  $InvFFZD = 1.0/FFZDelt$  ;
  - c. Set  $FFYDelt = Dum\_Real4$  ; delta-y (m)
  - d. Set  $InvFFYD = 1.0/FFYDelt$  ;
  - e. Set  $FFXDelt = Dum\_Real4$  ; delta-x (m)
  - f. Set  $NFFSteps = 2*Dum\_Int4$  ; number of time steps
  - g. Set  $MeanFFWS = Dum\_Real4$  ; mean full-field wind speed
  - h. Set  $InvMFFWS = 1.0/MeanFFWS$
  - i. Set  $FFDTime = FFXDelt/MeanFFWS$
  - j. Set  $FFRate = 1.0/FFDTime$
  - k. Skip through the next 5 variables (not used)
  - l. Set  $NZGrids = Dum\_Int4$
  - m. Set  $FFZHWid = 0.5*FFZDelt*( NZGrids - 1 )$  ; half the vertical size of the grid
  - n. Set  $NYGrids = Dum\_Int4$
  - o. Set  $FFYHWid = 0.5*FFYDelt*( NYGrids - 1 )$
  - p. IF ( $NFFComp == 3$ ) THEN Skip through the next 6 variables (not used)
6. IF  $TurbType == 7$  THEN: General Kaimal Model: skip the next 2 variables (not used)
7. ELSEIF  $TurbType == 8$  THEN: Mann model: skip the next 16 variables (not used)

## OUTPUT

$ErrStat : I$ ; the usual error status flag.

$TI(3) : R(ReKi)$ : Turbulence intensity for u, v, w.

It reads the NEW BLADED STYLE file and sets several variables belonging to the module ***FFWind***:  $NFFComp$ ,  $InvFFZD$ ,  $InvFFYD$ ,  $NFFSteps$ ,  $MeanFFWS$ ,  $InvMFFWS$ ,  $FFDTime$ ,  $FFRate$ ,  $NZGrids$ ,  $FFZHWid$ ,  $NYGrids$ ,  $FFYHWid$ .

## J4. READ\_BLADED\_GRIDS (UNWIND, CWISE, TI, ERRSTAT)

This subroutine continues reading *UnWind*, starting after the headers have been read. It reads the grids and converts the data to un-normalized wind speeds in m/s. Called by *FFInit* (*FFWind*).

### EXTERNAL (INVOKED) MODULES

From Parent Module *FFWind*[.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for reading wind file
<i>CWise</i>	L	IN	Clockwise rotation of turbine (Y=T)
<i>TI(3)</i>	R(ReKi)	IN	Turbulence intensities of the wind components as defined in the FF file, not necessarily the actual TI
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *NFFSteps* (*FFWind*))

### LOCAL VARIABLES

Name	Type	Value	Parameter/ Initialized	Description
<i>FF_Offset(3)</i>	R(ReKi) (/1.0,0.0,0.0/)PARAMETER			For "un-normalizing" the data
<i>CFirst</i>	I			Column index accounting for turbine rotation
<i>CLast</i>	I			Column index accounting for turbine rotation
<i>CStep</i>	I			+1 or -1 depending on turbine rotation
<i>Dum_Int2</i>	I(B2Ki)			Dummy var to read from file
<i>I</i>	I			Loop counter through 1-3 components
<i>IC</i>	I			Loop counter through columns
<i>IR</i>	I			Loop counter through rows
<i>IT</i>	I			Loop counter through time-steps
<i>TmpNumSteps</i>	I			<i>NFFStep</i> ( <i>FFWind</i> ) + 1

### STEPS

1. Write on the screen an informative message stating that a "*NYgrids* x *NZgrids* grid  $2^*FFYHWid$  meters wide, *GridBase* m to *GridBase*+ $2^*FFZHWid$  meters above ground with a characteristics wind speed of *MeanFFWS* m/s" (variables from *FFWind*)

2. Set  $TmpNumSteps = NFFSteps + 1$  (**FFWind**) ; this is a weird addition, but it is made so for the check in the loop below to see whether by chance the number of time points is odd or even. With this one, we think it is even.
3. ALLOCATE if .NOT. Allocated  $FFDATA(NZGrids,NYGrids,NFFComp,TmpNumSteps)$  (**FFWind**)
4. Initialize:  $FFData(:,:,,:)=0.0$
5. IF CWISE THEN: Set column indexing to account for direction of turbine rotation (**CWise**):
  - a. Set  $CFirst= NYGrids$  (**FFWind**);  $CLast= 1$ ;  $CStep= -1$
6. ELSE :
  - a. Set  $CFirst=1$ ;  $CLast= NYGrids$  ;  $CStep= 1$
7. Set  $NFFSteps = TmpNumSteps$  (**FFWind**)
8. DO\_LOOP  $IT=1, TmpNumSteps$  : Loop through all the time steps (possibly +1) and fill in  $FFDATA$  scaling data into m/s
  - a. DO\_LOOP  $IR=1, NZGrids$  (**FFWind**): Loop through all the rows
    - i. DO\_LOOP  $IC=CFirst, CLast, Cstep$ : Loop through all the columns
      1. DO\_LOOP  $I=1, NFFComp$  (**FFWind**): Loop through the U, V, W components
        - a. Read *Dum\_Int2* from *UnWind* file unit
        - b. IF *ErrStat*>0 THEN:
          - i. IF  $IT= TmpNumSteps$  THEN (at the end):
            1.  $NFFSteps= TmpNumSteps - 1$  (i.e. there really were an even number of steps)
            2. Set *ErrStat*=0
            3. EXIT LOOP
          - ii. ELSE warn user and RETURN with *ErrStat*=1
        - c. ELSE: Set  $FFData(IR,IC,I,IT) = MeanFFWS*(FF_Offset(I)+0.00001*TI(I)*Dum_Int2)$
      2. END\_LOOP I
    - ii. END\_LOOP IC
    - b. END\_LOOP IR
  9. END\_LOOP IT

## OUTPUT

*ErrStat* : I; the usual error status flag.

It reads the BLADED STYLE file dataset and sets  $FFData$  (**FFWind**) wind velocity components from BLADED format file.

## J5. READ\_SUMMARY\_FF(UNWIND, FILENAME, CWISE, ZCENTER, TI, ERRSTAT)

Subroutine to read the text summary file associated with the BLADED style date file to get normalizing parameters, the location of the grid, and the direction the grid was written to the binary file. Called by *FFInit (FFWind)*.

### EXTERNAL (INVOKED) MODULES

From Parent Module *FFWind*[.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for the file to open
<i>FileName</i>	C(*)	IN	Name of the summary file
<i>CWise</i>	L	OUT	Turbine rotation (for reading the order of the binary data)
<i>ZCenter</i>	R(ReKi)	OUT	Height at the center of the grid
<i>TI(3)</i>	R(ReKi)	OUT	Turbulence intensities of the wind components as defined in the FF file, not necessarily the actual TI
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *RefHt (FFWind)*)

### LOCAL VARIABLES

Name	Type	Value	Parameter/ Initialized	Description
<i>ZGOffset</i>	R(ReKi)			The vertical offset of the turbine on rectangular grid (allows turbulence not centered on turbine hub)
<i>NumStrings</i>	I	5	PARAMETER	Number of strings to look for in the file
<i>I</i>	I			Loop counter
<i>FirstIndx</i>	I			The first character of a line where data is
<i>LastIndx</i>	I			The last character of a line where data is
<i>LineCount</i>	I			Number of lines read in the file
<i>Status</i>	I			Status from I/O calls
<i>StrNeeded(NumStrings)</i>	L			Whether the string has been found
<i>LINE</i>	C(1024)			1 line string from the file

## STEPS

1. Initialize some variables:
  - a. *ErrStat* = 0;
  - b. *LineCount* = 0;
  - c. *StrNeeded*(:) = .TRUE.
  - d. *ZGOffset*=0.0
  - e. *RefHt*=0.0 (*FFWind*)
2. Open summary file: Call *OpenFInpFile* ( *UnWind*, TRIM( *FileName* ), *ErrStat* ) (NWTC\_IO within NWTC\_Library package)
3. DO WHILE ( ( *ErrStat* == 0 ) .AND. *StrNeeded*(NumStrings) )
  - a. *LineCount*=*LineCount*+1
  - b. Read *LINE* from *UnWind*
  - c. IF *ErrStat* <>0 THEN
    - i. IF ( *StrNeeded*(NumStrings-1) ) THEN warn user and RETURN with *ErrStat* = NumStrings+1 ELSE EXIT LOOP
  - d. Convert *LINE* to uppercase
  - e. IF *StrNeeded*(1) THEN : Get the rotation direction, using the string "CLOCKWISE":
    - i. Look for string "CLOCKWISE" in *LINE*, IF exists THEN:
      1. Look for a True/False value or 'Y' or 'N' and set *CWise* accordingly
      - ii. Set *StrNeeded*(1) = .FALSE.
  - f. ELSEIF *StrNeeded*(2) THEN : Get the hub height, using the strings "ZHUB" or "HUB HEIGHT":
    - i. Look for strings "ZHUB" or "HUB HEIGHT" in *LINE*, IF exists THEN:
      1. Read *RefHt* from *LINE*: READ ( *LINE*, \*, IOSTAT = *Status* ) *RefHt*
      2. IF *Status*<>0 THEN warn user set *ErrStat* = 2 and RETURN
      3. Set *StrNeeded*(2) = .FALSE.
  - g. ELSEIF *StrNeeded*(4) THEN : Get the mean wind speed "UBAR" and turbulence intensities from following lines for scaling BLADED-style FF binary files:
    - i. Look for string "UBAR" in *LINE*, IF exists THEN:
      1. Read *MeanFFWS* from *LINE*:
      2. IF *Status*<>0 THEN warn user set *ErrStat* = 4 and RETURN
      3. DO\_LOOP *I*=1,3 :
        - a. *LineCount*=*LineCount*+1
        - b. Read *LINE* from *UnWind*; IF problem with the read (*Status*<>0) THEN: warn user, Set *ErrStat*=*Status* and RETURN
        - c. Read *Ti*(*I*) numbers from *LINE* between "=" and "%" (or EOL); IF problems with the Read THEN warn user and RETURN with *ErrStat* = 4
      4. END\_LOOP
    - ii. Set *StrNeeded*(4) = .FALSE.

- h. ELSEIF *StrNeeded*(5) THEN: Get the grid "HEIGHT OFFSET", if it exists (in TurbSim). Otherwise, assume it's zero;  $ZGOffset = HH - GridBase - FFZHWid$ 
  - i. Look for string "HEIGHT OFFSET" in *LINE*, IF exists THEN:
    1. Read *ZGOffset* from *LINE* starting at "=" character till end of line; IF problem with the read (*Status*<>0) THEN: warn user, Set *ErrStat*=5 and RETURN
    2. Set *StrNeeded*(5) = .FALSE.
- 4. END\_LOOP
- 5. Set *ErrStat* = 0 and Close file unit *UnWind*
- 6. Set height of grid center:  $ZCenter = RefHt - ZGOffset$
- 7. Close(*Unwind*), i.e. the *FileName* file

## **OUTPUT**

*ErrStat*: I; the usual error flag

*CWise*: L; flag that establishes whether the turbine is rotating clockwise seen from upwind

*ZCenter*: R(ReKi); the height at the center of the grid.

*TI*(3): R(ReKi); turbulence intensity for the three components of wind velocity.

It sets other variables declared in parent module (*FFWind*): *RefHt*, *MeanFFWS*.

## J6. READ\_FF\_TOWER(UNWIND, WINDFILE, ERRSTAT)

Subroutine to read the binary tower file that corresponds with the BLADED-style FF binary file. The FF grid must be read before this subroutine is called (many checks are made to ensure the files belong together). Called by *FFInit (FFWind)*.

### EXTERNAL (INVOKED) MODULES

From Parent Module *FFWind*[.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for the wind file to open
<i>WindFile</i>	C(*)	IN	Name of the binary TurbSim file
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Ref\_Ht (FFWind)*)

### LOCAL VARIABLES

Name	Type	Value	Parameter/ Initialized	Description
<i>Dum_Real4</i>	R(SiKi)			Dummy 4-byte real number
<i>Dum_Int2</i>	I(B2Ki)			Dummy 2-byte integer
<i>Dum_Int4</i>	I(B4Ki)			Dummy 4-byte integer
<i>IC</i>	I			Loop counter for wind components
<i>IT</i>	I			Loop counter for time steps
<i>IZ</i>	I			Loop counter for z
<i>TOL</i>	R(ReKi)	1E-4	PARAMETER	Tolerance for wind file comparisons
<i>TI(3)</i>	R(SiKi)			Scaling values for "un-normalizing the data" [approx. turbulence intensities of the wind components]
<i>FF_Offset(3)</i>	R(ReKi)	(/1.0,0.,0.)	PARAMETER	Offset for "un-normalizing" data

### STEPS

1. Initialize *NTgrids=0 (FFWind)*
2. IF *NFFComp<>3 (FFWind)* THEN warn user and RETURN with *ErrStat=1*, since tower binary file requires 3 components of wind velocity

3. Open file: Call *OpenBInpFile* ( *UnWind*, TRIM(*WindFile*), *ErrStat*)\_\_(*NWTC\_IO* within *NWTC\_Library* package)
4. Read Header INFO and check that the tower file matches the binary wind file through some basic comparisons with variables read from BLADED binary file and established in *FFWind*:
  - a. Read *Dum\_Real4* from *UnWind* file (IF *ErrStat* <>0 user is warned and RETURN): this is grid delta-z (see *Read Bladed FF Header[0/1]*)
  - b. IF (ABS(*Dum\_Real4*\*InvFFZD-1)>TOL THEN warn user of non-match and RETURN with *ErrStat*=1
  - c. Read *Dum\_Real4* from *UnWind* file (IF *ErrStat* <>0 user is warned and RETURN): this is grid delta-x
  - d. IF (ABS(*Dum\_Real4*\*InvMFFWS/FFDTime-1)>TOL THEN warn user of non-match and RETURN with *ErrStat*=1
  - e. Read *Dum\_Real4* from *UnWind* file (IF *ErrStat* <>0 user is warned and RETURN): this is grid Zmax
  - f. IF (ABS(*Dum\_Real4*/GridBase-1)>TOL THEN warn user of non-match and RETURN with *ErrStat*=1
  - g. Read *Dum\_Int4* from *UnWind* file (IF *ErrStat* <>0 user is warned and RETURN): this is grid NumZ (number of points along Z)
  - h. Set *NTgrids*=*Dum\_Int4*
  - i. Read *Dum\_Real4* from *UnWind* file (IF *ErrStat* <>0 user is warned and RETURN): this is UHub
  - j. IF (ABS(*Dum\_Real4*\*InvMFFWS -1)>TOL THEN warn user of non-match and RETURN with *ErrStat*=1 and *NTgrids*=0
  - k. DO\_LOOP *IC*=1,3: read wind velocity component *Tis*: Read *TI*(*IC*) from *UnWind*: IF *ErrStat*<>0 warn user and RETURN with *NTgrids*=0
5. ALLOCATE IF *NTgrids*>0 and .NOT. Allocated : *FFtower*( *NFFComp*, *NTgrids*, *NFFSteps* ) (*FFWind*) (the routine shows signs of missing parts here, no big deal however, **minor bug**)
6. DO\_LOOP *IT*=1, *NFFSteps*: Read the 16-bit data and scale it to 32-bit; time step loop
  - a. DO\_LOOP *IZ*=1, *NTgrids* ; vertical spacing loop
    - i. DO\_LOOP *IC*=1, *NFFComp* ; loop on wind velocity components
      1. Read *Dum\_Int2* from *UnWind* (RETURN with *ErrStat*=1 and *NTgrids*=0 IF *ErrStat*<>0 and warn user)
        - a. Set *FFtower*(*IC*,*IZ*,*IT*) = MeanFFWS\*(*FF\_Offset*(*IC*) + 0.00001 \* *TI*(*IC*) \* *Dum\_Int2*); wind-component scaled to m/s
      - ii. END\_LOOP
    - b. END\_LOOP
  7. END\_LOOP
  8. Close(*Unwind*), i.e. the *WindFile* file
  9. Generate a message stating that “*NFFSteps* time steps of *NTgrids* x1 tower data grids were processed”

## OUTPUT

*ErrStat: I;* the usual error flag

It reads the BLADED STYLE tower data file and sets *FFtower (FFWind)* wind velocity components.

## J7. READ\_TURBSIM\_FF(UNWIND, WINDFILE, ERRSTAT)

Subroutine to read the binary TurbSim-format FF file (.bts). It fills the FFData array with velocity data for the grids and fills the *FFtower* array with velocities at points on the tower (if data exists). Called by *FFInit* (*FFWind*).

### EXTERNAL (INVOKED) MODULES

From Parent Module *FFWind*[.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for the wind file to open
<i>WindFile</i>	C(*)	IN	Name of the binary TurbSim file
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Ref\_Ht* (*FFWind*))

### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>Dum_Real4</i>	R(SiKi)			Dummy 4-byte real number
<i>Dum_Int1</i>	I(B1Ki)			Dummy 1-byte integer
<i>Dum_Int2</i>	I(B2Ki)			Dummy 2-byte integer
<i>Dum_Int4</i>	I(B4Ki)			Dummy 4-byte integer
<i>IC</i>	I			Loop counter for wind components
<i>IT</i>	I			Loop counter for time steps
<i>IY</i>	I			Loop counter for y
<i>IZ</i>	I			Loop counter for z
<i>NChar</i>	I			Number of characters in the description string
<i>VSlope(3)</i>	R(SiKi)			Slope for "un-normalizing" data
<i>VOffset (3)</i>	R(SiKi)			Offset for "un-normalizing" data
<i>DescStr</i>	C(1024)			Description string contained in the file

### STEPS

1. Set *NFFComp* =3 (*FFWind*)

2. Open file: Call ***OpenBInpFile*** ( *UnWind*, TRIM( *FileName* ), *ErrStat*)\_(**NWTC\_IO** within **NWTC\_Library** package)
3. Read Header INFO:
  - a. Skip first data (Read *Dum\_Int2* from *UnWind* file (IF *ErrStat* <>0 user is warned and RETURN): File ID)
  - b. Multiple instances of Read *Dum\_Int4* from *UnWind* file (IF *ErrStat* <>0 user is warned and RETURN) to read other variables needed in **FFWind**:
    - i. Set *NZgrids* = *Dum\_Int4*; the number of grid points vertically
    - ii. Set *NYgrids* = *Dum\_Int4*; the number of grid points laterally
    - iii. Set *NTgrids* = *Dum\_Int4*; number of tower points
    - iv. Set *NFFSteps* = *Dum\_Int4*; number of time steps
    - v. Set *InvFFZD* =  $1.0 / Dum\_Real4$ ; 1/dz=1/grid spacing in vertical direction
    - vi. Set *FFZHWid* =  $0.5 * (NZgrids - 1) * Dum\_Real4$ ; half the grid height
    - vii. Set *InvFFYD* =  $1.0 / Dum\_Real4$ ; 1/dy=1/grid spacing in lateral direction
    - viii. Set *FFYHWid* =  $0.5 * (NYgrids - 1) * Dum\_Real4$ ; half the grid width
    - ix. Set *FFDTime* = *Dum\_Real4*; grid time spacing
    - x. Set *FFRate* =  $1.0 / FFDTime$
    - xi. Set *MeanFFWS* = *Dum\_Real4*; Mean Wind Speed at hub height
    - xii. Set *InvMFFWS* =  $1.0 / MeanFFWS$ ;
    - xiii. Set *RefHt* = *Dum\_Real4*; reference (or hub) height
    - xiv. Set *GridBase* = *Dum\_Real4*; height of the bottom of the grid
    - xv. DO *IC*=1,*NFFComp*: Read scaling factors
      1. Read *VSlope*(*IC*) and *VOffset*(*IC*) from *UnWind* (IF *ErrStat* <>0 THEN warn user and RETURN )
    - xvi. Set *nchar* = *Dum\_Int4*; number of characters in the description string, max 200
    - xvii. Initialize *DescStr*='' then
    - xviii. DO *IC*=1,*nchar*: Read description string in ASCII integer representation
      1. READ (*UnWind*, IOSTAT=*ErrStat*) *Dum\_Int1* (RETURN incase *ErrStat*<>0 and warn user)
      2. *DescStr*(*IC*:*IC*) = ACHAR(*Dum\_Int1* )
      3. IF *DescStr* is too long then warn user and EXIT LOOP
    - xix. END\_LOOP
  10. Generate an information message stating that a "NYgrids x NZgrids grid  $2 * FFYHWid$  meters wide, *GridBase* m to *GridBase*+ $2 * FFZHWid$  meters above ground with a characteristics wind speed of *MeanFFWS* m/s" (variables from **FFWind**)
  11. ALLOCATE if .NOT. Allocated: *FFDATA*(*NZGrids*,*NYGrids*,*NFFComp*,*NFFSteps*) (**FFWind**)
  12. ALLOCATE IF *NTgrids*>0 and .NOT. Allocated : *FFtower*( *NFFComp*, *NTgrids*, *NFFSteps* ) (**FFWind**)
  4. DO\_LOOP *IT*=1, *NFFSteps*: Read the 16-bit data and scale it to 32-bit; time step loop
    - a. DO\_LOOP *IZ*=1, *NZgrids* ; vertical spacing loop

- i. DO\_LOOP IY=1, NYgrids ; lateral spacing loop
  - 1. DO\_LOOP IC=1, NFFComp ; loop on wind velocity components
    - a. Read Dum\_Int2 from UnWind (RETURN incase ErrStat<>0 and warn user)
    - b. Set  $FFData(IZ,IY,IC,IT) = ( Dum_{Int2} - Voffset(IC) ) / VSlope(IC);$  wind-component scaled to m/s
  - 2. END\_LOOP
- ii. END\_LOOP
- b. END\_LOOP
- c. DO\_LOOP IZ=1, NTgrids: Read the tower data for the current time step
  - i. DO\_LOOP IC=1, NFFComp ; loop on wind velocity components
    - 1. Read  $Dum_{Int2}$  from  $UnWind$  (RETURN incase ErrStat<>0 and warn user)
    - 2. Set  $FFtower(IC,IZ,IT) = ( Dum_{Int2} - Voffset(IC) ) / VSlope(IC);$  wind-component scaled to m/s
  - ii. END\_LOOP
- 5. END\_LOOP
- 6. Close( $Unwind$ ), i.e. the *WindFile* file
- 7. Generate a message stating that “ $NFFSteps$  time steps of  $FFRate$  – Hz full-field data  $FFDTime*(NFFSteps-1)$  seconds were processed”

## OUTPUT

*ErrStat: I;* the usual error flag

It reads the TurbSim STYLE file dataset and sets *FFData* and *FFtower* (*FFWind*) wind velocity components.

It reads and sets several variables belonging to the module ***FFWind***: *NFFComp*, *InvFFZD*, *InvFFYD*, *NFFSteps* , *MeanFFWS*, *InvMFFWS*, *FFDTime*, *FFRate*, *NZGrids*, *FFZHWid*, *NYGrids*, *NTgrids*, *FFYHWid*, *GridBase*, *Ref\_Ht*.

## J8. FF\_GETWINDSPEED (TIME, INPUTPOSITION, ERRSTAT)

This function returns the velocities at the specified time and space. It determines if the point is on the FF grid or tower points and calls the corresponding interpolation routine, which returns the velocities at the specified time and space. Called by WindInf GetVelocity (InflowWind).

### EXTERNAL (INVOKED) MODULES

From Parent Module *FFWind*[.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>Time</i>	R(ReKi)	IN	Time
<i>InputPosition (3)</i>	R(ReKi)	IN	Position (X, Y, Z)
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Initialized (FFWind)*)

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>TOL</i>	R(ReKi)	1E-3	PARAMETER	NOT USED

#### STEPS

1. Make sure that *FFWind* module is initialized:
  - a. IF .NOT. *Initialized (FFWind)* THEN:
    - i. warn user on the screen
    - ii. Set *ErrStat*=1
    - iii. RETURN
  - b. ELSE Set *ErrStat*=0
2. Find out if the location is on the grid OR on tower points; interpolate and return the value:  
 $FF\_GetWindSpeed\%Velocity = \underline{FF\_Interp}(Time, InputPosition, ErrStat) \underline{(FFWind)}$

#### OUTPUT

*ErrStat*: I; the usual error flag.

*FF\_GetWindSpeed*: TYPE(**InflIntrpOut**) (*SharedInflowDefns*); the wind velocity (no tower effect) at the point of interest following the FF file type (either Bladed or TurbSim).

## J9. FF\_INTERP (TIME, POSITION, ERRSTAT)

This function is used to interpolate into the full-field wind array (or tower array if it has been defined and is necessary for the given inputs).

Called by *FF\_GetWindSpeed (FFWind)*.

### THEORY

It receives X, Y, Z and TIME from the calling routine. It then computes a time shift due to a nonzero X based upon the average wind speed. At time=0, a point half the grid width downstream (FFYHWid) will index into the zero time slice. If we did not do this, any point downstream of the tower at the beginning of the run would index outside of the array. This assumes the grid width is at least as large as the rotor. If it is not then the interpolation will not work.

The modified time is used to decide which pair of time slices to interpolate within and between. After finding the two time slices, it decides which four grid points bound the (Y,Z) pair. It does a bilinear interpolation for each time slice. Linear interpolation is then used to interpolate between time slices. This routine assumes that X is downwind, Y is to the left when looking downwind and Z is up. It also assumes that no extrapolation will be needed.

If tower points are used, it assumes the velocity at the ground is 0. It interpolates between heights and between time slices, but ignores the Y input.

### EXTERNAL (INVOKED) MODULES

From Parent Module *FFWind.f90*

#### INPUT

Name	Type/Module	INTENT	Description
<i>Time</i>	R(ReKi)	IN	Time
<i>Position (3)</i>	R(ReKi)	IN	Position (X, Y, Z)
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *InvMFFWS (FFWind)*)

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>TOL</i>	R(ReKi)	1E-3	PARAMETER	Tolerance for determining if 2 reals are the same (extrapolation)
<i>TimeShifted</i>	R(ReKi)			Shifted time (necessary because we're keeping x constant)
<i>W_YH_Z</i>	R(ReKi)			Temp. value for interpolation along z at high y

<i>W_YH_ZH</i>	R(ReKi)	Value at high z and high y (interpolation corner)
<i>W_YH_ZL</i>	R(ReKi)	Value at low z and high y (interpolation corner)
<i>W_YL_Z</i>	R(ReKi)	Temp. value for interpolation along z at low y
<i>W_YL_ZH</i>	R(ReKi)	Value at high z and low y (interpolation corner)
<i>W_YL_ZL</i>	R(ReKi)	Value at high z and low y (interpolation corner)
<i>TGRID</i>	R(ReKi)	Fractional distance between time grids.
<i>Y</i>	R(ReKi)	Temp value to find index in grid along y
<i>YGRID</i>	R(ReKi)	Fractional distance between grids in the y direction.
<i>Z</i>	R(ReKi)	Temp. value for interpolation
<i>ZGRID</i>	R(ReKi)	Temp. value to find index in grid along z
<i>Wnd</i> (2)	R(ReKi)	Temp. array for interpolation at upper and lower time slice
<i>IDIM</i>	I	Counter
<i>IG</i>	I	Counter for lower or upper time slice bounding the current TIME
<i>IT</i>	I	counter
<i>ITHI</i>	I	High index for tower points
<i>ITLO</i>	I	Low index for tower points
<i>IYHi</i>	I	High Index for lateral (y) points
<i>IYLo</i>	I	Low Index for lateral (y) points
<i>IZHI</i>	I	High Index for lateral (y) points
<i>IZLO</i>	I	Low Index for lateral (y) points
<i>OnGrid</i>	L	Whether or not we are on grid: False=tower point

**STEPS**

1. Initialize: *FF\_Interp(:)=0.0* and *Wnd(:)=0.0*
2. Perform the time shift: *TimeShifted = Time+ ( FFYHWid - Position(1) )\*InvMFFWS (FFWind)*: here it is assumed that the turbulence is moving at MCTWS speed, and therefore the x-position is translated into a new time based on that translational velocity

3. Set  $TGRID = TimeShifted * FFRate (FFWind)$ :
4. Find the “time slices” that bound the shifted time so to linearly interpolate:
  - a. Set  $ITLO = \text{INT}(TGRID) + 1$
  - b. Set  $ITHI = ITLO + 1$
  - c.  $T = TGRID - (ITLO - 1)$ ; relative position between  $ITLO$  and  $ITHI$
  - d. IF  $ITLO \geq NFFSteps (FFWind)$ .OR.  $ITLO < 1$  THEN : check we are within bounds
    - i. IF  $ITLO == NFFSteps$  THEN
      1. Set  $ITHI = ITLO$
      2. IF  $T \leq TOL$  THEN (on the last point) Set  $T=0.0$  ELSE Set  $ITLO = ITHI - 1$
    - ii. ELSE warn user and RETURN with  $ErrStat=1$
5. Find the bounding rows for the Z position. [The lower-left corner is (1,1) when looking upwind.]:
  - a.  $ZGRID = (\text{Position}(3) - GridBase (FFWind)) * InvFFZD (FFWind)$
  - b. IF  $ZGRID > -TOL$  THEN
    - i. Set  $OnGrid = \text{TRUE}$ .
    - ii. Set  $IZLO = \text{INT}(ZGRID) + 1$
    - iii. Set  $IZHI = IZLO + 1$
    - iv.  $Z = ZGRID - (IZLO - 1)$
    - v. IF  $IZLO < 1$  THEN
      1. IF  $IZLO == 0$  .AND.  $Z \geq -TOL$  THEN Set  $Z=0.0$  and  $IZLO=1$  ELSE warn user and RETURN with  $ErrStat=1$
    - vi. ELSEIF  $IZLO \geq NZGrids (FFWind)$  THEN
      1. IF  $IZLO == NZGrids$  .AND.  $Z \leq TOL$  THEN Set  $Z=0.0$  and  $IZHI = IZLO$  ELSE warn user and RETURN with  $ErrStat=3$
  - c. ELSE (point on the tower)
    - i. Set  $OnGrid = \text{FALSE}$ .
    - ii. IF  $NTgrids < 1$  THEN warn user and RETURN with  $ErrStat=1$
    - iii. Set  $IZLO = \text{INT}(-ZGRID) + 1$
    - iv. IF  $IZLO \geq NTgrids$  THEN
      1. Set  $IZLO = NTgrids$
      2.  $Z = 1 - \text{Position}(3) / (GridBase - (IZLO - 1) / InvFFZD)$
    - v. ELSE Set  $Z = \text{ABS}(ZGRID) - (IZLO - 1)$
    - vi. Set  $IZHI = IZLO + 1$
6. IF  $OnGrid$  (i.e. no tower points) THEN: Find the bounding columns for the Y position (The lower-left corner is (1,1) when looking upwind).
  - a.  $YGRID = (\text{Position}(2) + FFYHWid) * InvFFYD (FFWind)$ ; normalized y-distance
  - b.  $IYLO = \text{INT}(YGRID) + 1$ ; low index
  - c.  $IYHI = IYLO + 1$ ; high index
  - d.  $Y = YGRID - (IYLO - 1)$ ; relative location between  $IYLO$  and  $IYHI$
  - e. IF  $IYLO \geq NYgrids (FFWind)$  .OR.  $IYLO < 1$  THEN
    - i. IF  $IYLO == 0$  .AND.  $Y \geq -TOL$  THEN

1. Set  $Y=0.0$
  2.  $IYLO=1$
  - ii. ELSEIF  $IYLO==NYGrids$  .AND.  $Y<=TOL$  THEN
    1. Set  $Y=0.0$
    2.  $IYHI=IYLO$
  - iii. ELSE warn user and RETURN with  $ErrStat=2$
  - f. DO\_LOOP  $IDIM=1,NFFComp$  (**FFWind**) : Interpolate for [U/V/W] component of wind velocity within the grid, by first interpolating along z and then along y (for the two time slices respectively); then interpolate in time:
    - i. Set  $IG=1$  (lower time slice)
    - ii. DO\_LOOP  $IT=ITLO,ITHI$ : find values of wind velocity at the four corners for the two time slices:
      1.  $W_YL_ZL=FFData(IZLO, IYLO, IDIM, IT)$  (**FFWind**); Lower Z & Lower L corner
      2.  $W_YL_ZH=FFData(IZHI, IYLO, IDIM, IT)$  ; Higher Z & Lower L corner
      3.  $W_YH_ZL=FFData(IZLO, IYHI, IDIM, IT)$  ; Lower Z & Higher L corner
      4.  $W_YH_ZH=FFData(IZHI, IYHI, IDIM, IT)$  ; Lower Z & Higher L corner
      5.  $W_YL_Z = ( W_YL_ZH - W_YL_ZL ) * Z + W_YL_ZL$ ; interpolate along z, at low Y
      6.  $W_YH_Z = ( W_YH_ZH - W_YH_ZL ) * Z + W_YH_ZL$ ; interpolate along z, at high Y
      7.  $Wnd(IG) = ( W_YH_Z - W_YL_Z ) * Y + W_YL_Z$ ; interpolate along y
      8.  $IG=IG+1$  (upper time slice)
    - iii. END\_LOOP on time slices
    - iv.  $FF_Interp(IDIM) = ( Wnd(2) - Wnd(1) ) * T + Wnd(1)$  ; interpolate between two time slices
  - g. END\_LOOP on components
7. ELSE (tower points)
- a. DO\_LOOP  $IDIM=1,NFFComp$  (**FFWind**) : Interpolate for [U/V/W] component of wind velocity first along z then along time
    - i. Set  $IG=1$  (lower time slice)
    - ii. DO\_LOOP  $IT=ITLO,ITHI$ : find values of wind velocity at the four corners for the two time slices:
      1.  $W_YH_ZL=FFTower( IDIM, IZLO, IT )$  (**FFWind**); Lower Z
      2. IF  $IZHI > NTGrids$  THEN  $W_YH_ZH = 0.0$  ; Higher Z
      3. ELSE  $W_YH_ZH=FFTower( IDIM, IZHI, IT )$  ; Higher Z
      4.  $Wnd(IG) = ( W_YH_ZH - W_YH_ZL ) * Z + W_YH_ZL$ ; interpolate along z
      5.  $IG=IG+1$  (upper time slice)
    - iii. END\_LOOP on time

iv.  $FF\_Interp(IDIM) = ( Wnd(2) - Wnd(1) ) * T + Wnd(1) ;$  interpolate between two time slices

b. END\_LOOP on components

## OUTPUT

*ErrStat: I;* the usual error flag.

*FF\_Interp(3): R(ReKi);* the wind velocity (no tower effect) at the point of interest following the FF file type (either BLADED or TurbSim).

## J10. FF\_GETRVALUE(RVARNOME,ERRSTAT)

Function that returns a real scalar value whose name is listed in the *VarName* input argument and that belongs to the module *FFWind*. If the name is not recognized, an error is returned in *ErrStat*. (Note it is called as *FF GetValue*, but in the *FFWind* module an interface is realized between *FF GetValue* and *FF GetRValue*). Called by *WindInf ADhack diskVel*, *WindInf ADhack DIcheck*, and *WindInf Init (InflowWind)*.

### EXTERNAL (INVOKED) MODULES

From Parent Module *FFWind*[.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>RVarName</i>	C(*)	IN	Variable name string
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Initialized (FFWind)*)

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>VarNameUC</i>	C(20)			Upper case <i>RVarName</i>

#### STEPS

1. IF .NOT. *Initialized* (parent module *FFWind*): module NOT initialized: THEN:
  - a. warn user and RETURN with *ErrStat*=1
2. ELSE initialize *ErrStat*=0
3. Transform to upper case *VarNameUC* = *RVarName*
4. Based on *VarNameUC*:
  - a. 'HUBHEIGHT' or' REFHEIGHT'  $\rightarrow$  *FF\_GetRValue* = *RefHt* (*FFWind*)
  - b. 'GRIDWIDTH' or 'FFYWID'  $\rightarrow$  *FF\_GetRValue* = *FFYHWid*\*2 (*FFWind*)
  - c. 'GRIDHEIGHT' or 'FFZWID'  $\rightarrow$  *FF\_GetRValue* = *FFZHWid*\*2 (*FFWind*)
  - d. 'MEANFFWS'  $\rightarrow$  *FF\_GetRValue* = *MeanFFWS* (*FFWind*)
  - e. All other cases: warn user of invalid variable name and set *ErrStat*=1

#### OUTPUT

*FF\_GetRValue*: R(ReKi); the value of the variable whose name string was input.

*ErrStat*: I; the usual error flag.

## J11. FF\_TERMINATE (ERRSTAT)

Subroutine to deallocate memory, and to un-set the initialization flag. Files (if open) are closed in *InflowWind*. Called by WindInf Terminate (InflowWind).

### EXTERNAL (INVOKED) MODULES

From Parent Module *FFWind*[.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Initialized (FFWind)*)

### LOCAL VARIABLES

N/A

### STEPS

1. Set *ErrStat*=0
2. DEALLOCATE if allocated :
  - a. *FFData, FFTower (FFWind)*
3. Set *Initialized* = .FALSE.; initialization flag: if T=initialized

### OUTPUT

*ErrStat*: I; the usual error flag

It deallocates memory.

## APPENDIX K. FDWIND MODULE

### GENERAL ORGANIZATION

This module reads and processes 4-dimensional wind fields. The subroutines were originally created by Marshall Buhl to read LES data provided by researchers at NCAR. It was later updated by Bonnie Jonkman to read DNS data provided by researchers at CoRA. Data is assumed in m/s.

### EXTERNAL MODULES

- *NWTC\_Library* [see NWTC\_Library.f90]
- *SharedInflowDefns* [see SharedInflowDefs.f90]

### PRIVATE (LOCALLY DECLARED) DATA

Name	Type	Value	Parameter/ Initialized	Description
<i>DelXgrid</i>	R(ReKi)			The non-dimensional distance between grid points in the x direction.
<i>DelYgrid</i>	R(ReKi)			The non-dimensional distance between grid points in the y direction.
<i>DelZgrid</i>	R(ReKi)			The non-dimensional distance between grid points in the z direction.
<i>FDper</i>	R(ReKi)			Total time in dataset.
<i>FDTIME</i> (2)	R(ReKi)			Times for the 4D wind files.
<i>FDu</i> (:,:,:,:)	R(ReKi)			( <i>Num4Dx</i> , <i>Num4Dy</i> , <i>Num4Dz</i> , <i>Num4Dt</i> ), u-component array.
<i>FDv</i> (:,:,:,:)	R(ReKi)			( <i>Num4Dx</i> , <i>Num4Dy</i> , <i>Num4Dz</i> , <i>Num4Dt</i> ), v-component array.
<i>FDw</i> (:,:,:,:)	R(ReKi)			( <i>Num4Dx</i> , <i>Num4Dy</i> , <i>Num4Dz</i> , <i>Num4Dt</i> ), w-component array.
<i>FDuData</i> (:,:,:,:)	R(ReKi)			( <i>Num4Dx</i> , <i>Num4Dy</i> , <i>Num4Dz</i> , <i>Num4Dt</i> ), u-component array with advection.
<i>FDvData</i> (:,:,:,:)	R(ReKi)			( <i>Num4Dx</i> , <i>Num4Dy</i> , <i>Num4Dz</i> , <i>Num4Dt</i> ), v-component array with advection.
<i>FDwData</i> (:,:,:,:)	R(ReKi)			( <i>Num4Dx</i> , <i>Num4Dy</i> , <i>Num4Dz</i> , <i>Num4Dt</i> ), w-component array with advection.
<i>Lx</i>	R(ReKi)			Fractional location of tower centerline, from

			upwind end to downwind end of the dataset.
<i>Ly</i>	R(ReKi)		Fractional location of tower centerline from right (looking downwind) to left side of the dataset.
<i>Lz</i>	R(ReKi)		Fractional location of hub height from bottom to top of dataset.
<i>OFDsets</i> (3)	R(ReKi)		OFDsets to convert integer data to actual wind speeds.
<i>PrevTime</i>	R(ReKi)	SAVE	The previous time this was called -- so we can go back in time if necessary
<i>RotDiam</i>	R(ReKi)		Rotor diameter.
<i>ScalFact</i> (3)	R(ReKi)		Scaling factors to convert integer data to actual wind speeds.
<i>ScaleVel</i>	R(ReKi)		Scaling velocity, U0. 2*U0 is the diFDerence in wind speed between the top and bottom of the wave.
<i>Times4D</i> (:)	R(ReKi)		The list of times for the 4D-wind input files.
<i>Tm_max</i>	R(ReKi)		Total non-dimensional time of the dataset.
<i>TSclFact</i>	R(ReKi)		Scale factor for time (h/U0).
<i>T_4D_En</i>	R(ReKi)		Time at which the wave event ends.
<i>T_4D_St</i>	R(ReKi)		Time at which the wave event starts.
<i>Xmax</i>	R(ReKi)		The dimensional downwind length of the dataset.
<i>Xt</i>	R(ReKi)		Distance of the tower from the upwind end of the dataset.
<i>Ymax</i>	R(ReKi)		The dimensional lateral width of the dataset.
<i>Yt</i>	R(ReKi)		Distance of the tower from the right side of the dataset (looking downwind).
<i>Zmax</i>	R(ReKi)		The dimensional vertical height of the

			dataset.	
Zt	R(ReKi)		Distance of the hub from the bottom of the dataset.	
Zref	R(ReKi)		The reference height (hub height)	
FD_DF_X	I		The decimation factor for the 4D wind data in the x direction.	
FD_DF_Y	I		The decimation factor for the 4D wind data in the y direction.	
FD_DF_Z	I		The decimation factor for the 4D wind data in the z direction.	
FDFileNo	I		The 4D wind file number.	
FDRecL	I		The length, in bytes, of the LE binary records.	
Ind4DAdv	I		Index of the file to be used in advection	
Ind4Dnew	I		Index of the newest 4D wind file.	
Ind4Dold	I		Index of the older 4D wind file.	
Num4Dt	I		The number of 4D wind grids, one grid per time step.	
Num4DtD	I	2	PARAMETER	The number of 4D wind grids stored in memory, normally 2
Num4Dx	I		The number of 4D wind grid points in the x direction.	
Num4DxD	I		The decimated number of 4D wind grid points in the x direction.	
Num4DxD1	I		The decimated number of 4D wind grid points in the x direction minus 1.	
Num4Dy	I		The number of 4D wind grid points in the y direction.	
Num4DyD	I		The decimated number of 4D wind grid points in the y direction.	

<i>Num4DyD1</i>	I	The decimated number of 4D wind grid points in the y direction minus 1.
<i>Num4Dz</i>	I	The number of 4D wind grid points in the z direction.
<i>Num4DzD</i>	I	The decimated number of 4D wind grid points in the z direction.
<i>Num4DzD1</i>	I	The decimated number of 4D wind grid points in the z direction minus 1.
<i>NumAdvect</i>	I	Number of frozen timesteps to advect past the turbine
<i>Shft4Dnew</i>	I	Number of times the x-data needs to be shifted for advection
<i>Times4DIx ()</i>	I	Index number of the 4D time files (used for advection)
<i>FDUnit</i>	I	Unit number for reading wind files
<i>Advect</i>	L	Flag to indicate whether or not to advect a given data set or to just use the time step files
<i>VertShft</i>	L	Flag to indicate whether or not to shift the z values for the w component.
<i>Initialized</i>	L .FALSE.	Initialized, SAVE
<i>AdvFiles ()</i>	C(5)	
<i>FDSpath</i>	C(1024)	The path to the 4D wind files.

- New PRIVATE Routines and Functions

Name	Description
<u><i>ReadFDP</i></u>	It reads input parameters for the LES
<u><i>Read4DData</i></u>	It reads 1 time-step's worth of LES wind data for one component
<u><i>ReadAll4DData</i></u>	It reads data into an array

<u>Read4Dtimes</u>	It reads time array for the 4D data
<u>LoadLESData</u>	It reads binary data from U, V, W files and stores them in FD[u/v/w]
<u>Load4DData</u>	It reads data from storage array $FD[u/v/w]Data$ (used when ADVECT=.TRUE.) and populates main array $FD[u/v/w]$ for the time step specified

**PUBLIC (LOCALLY DECLARED) DATA**

- New PUBLIC Routines and Functions

Name	Description
<u>FD_Init</u>	Initialize module and reads in the wind data files
<u>FD_GetWindSpeed</u>	Interpolates the wind velocity components and returns them for the points and time of interest form the wind data file
<u>FD_GetValue</u>	Return values of requested scalar variables
<u>FD_Terminate</u>	Terminates, closes file, and deallocates module

## K1. FDINIT (UNWIND, WINDFILE, REFHT, ERRSTAT)

Subroutine called at the beginning of a simulation by WindInf Init (*InflowWind*). It reads the 4D wind file (.fdp).

### EXTERNAL (INVOKED) MODULES

From Parent Module *FDWind*[.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	unit number for reading wind files
<i>WindFile</i>	C(*)	IN	Name of the 4D wind parameter file (.fdp)
<i>RefHt</i>	R(ReKi)	IN	The reference height for the billow (should be hub height)
<i>ErrStat</i>	I	OUT	If $<>0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Initialized (FDWind)*)

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>FDTsfile</i>	C(1024)			Name of the 4D time step file
<i>FDTimStp</i>	R(ReKi)			Average time step for 4D wind data
<i>IT</i>	I			Counter

#### STEPS

1. IF *Initialized* (parent module *FDWind*): module already initialized: THEN:
  - a. warn user and RETURN with *ErrStat*=1
2. ELSE *ErrStat*=0 and Call NWTC\_Init (*NWTC\_Library*)
3. Set *ZRef* = *RefHt*
4. Read the main 4D input file: Call ReadFDP( *UnWind*, *WindFile*, *FDTsfile*, *ErrStat* ) (*FDWind*)
5. Read time array, which must be scaled and shifted later using *TSclFact* and *T\_4D\_St* (*FDWind*): Call Read4Dtimes(*UnWind*, *FDTsfile*, *ErrStat*) (*FDWind*)
6. Calculate some variables declared in *FDWind* based on other variables read from the previous call and also declared in *FDWind*:
  - a.  $FDRecL = 2 * Num4Dx * Num4Dy$  ; The length, in bytes, of the 4D binary records.
  - b.  $Num4DxD = ( Num4Dx + FD_DF_X - 1 ) / FD_DF_X$  ; The decimated number of 4D wind grid points in the x direction.

- c.  $Num4DyD = ( Num4Dy + FD\_DF\_Y - 1 )/FD\_DF\_Y;$  The decimated number of 4D wind grid points in the y direction.
  - d.  $Num4DzD = ( Num4Dz + FD\_DF\_Z - 1 )/FD\_DF\_Z;$  The decimated number of 4D wind grid points in the z direction.
  - e.  $Num4DxD1 = Num4DxD - 1;$  The decimated number of 4D wind grid points in the x direction minus 1.
  - f.  $Num4DyD1 = Num4DyD - 1;$  The decimated number of 4D wind grid points in the y direction minus 1.
  - g.  $Num4DzD1 = Num4DzD - 1;$  The decimated number of 4D wind grid points in the z direction minus 1.
  - h.  $Tm\_max = Times4D(Num4Dt);$  Time of end of dataset.
  - i. IF ADVECT (**FDWind**) THEN
    - i.  $FDTimStp = Xmax / (( Num4Dx - 1 ) * ( ScaleVel ) * Num4Dt);$  avg. time step
    - ii.  $FDper = FDTimStp * Num4Dt;$  Total time in dataset. (We have periodic time, so multiply by number of time steps, without subtracting 1)
    - iii.  $TSclFact = FDper / Tm\_max;$  Equivalent scale factor for time.
  - j. ELSE:
    - i.  $FDper = TSclFact * Tm\_max;$  Total time in dataset
    - ii.  $FDTimStp = FDper / ( Num4Dt - 1 );$  avg. time step
  - k.  $T\_4D\_En = T\_4D\_St + FDper;$  Time for the end of the dataset.
  - l.  $Xt = Xmax * Lx;$  Distance of the tower from the upwind end of the dataset.
  - m.  $Yt = Ymax * Ly;$  Distance of the tower from the right side of the dataset (looking downwind).
  - n.  $Zt = Zmax * Lz;$  Distance of the hub from the bottom of the dataset.
  - o.  $DelXgrid = 1.0 / Num4DxD1;$  The nondimensional distance between grid points in the x direction.
  - p.  $DelYgrid = 1.0 / Num4DyD1;$  The nondimensional distance between grid points in the y direction.
  - q.  $DelZgrid = 1.0 / Num4DzD1;$  The nondimensional distance between grid points in the z direction.
7. DO\_LOOP IT=1,Num4Dt : Scale and shift the times array
- a.  $Times4D(IT) = TSclFact * Times4D(IT) + T\_4D\_St (\textbf{FDWind})$
8. END\_LOOP
9. ALLOCATE IF .NOT. Allocated the following **FDWind** arrays:  
 $FD[u/v/w](Num4DxD, Num4DyD, Num4DzD, 2)$
10. IF ADVECT THEN:
- i. ALLOCATE  $FD[u/v/w]Data (Num4DxD, Num4DyD, Num4DzD, Num4Dt)$  (**FDWind**)
  - ii. Call **ReadAll4DData**(UnWind, ErrStat)(**FDWind**)
11. Set  $Ind4Dold = 1$  and  $Ind4Dnew = 2$  and  $Shft4Dnew = 0$  (**FDWind**)

12. IF  $T\_4D\_St \geq 0.0$  THEN  $FDFileNo=1$  (not sure what this is)
13. ELSE:
  - a.  $FDFileNo=Num4Dt$  (***FDWind***)
  - b. DO\_LOOP  $IT=1, Num4Dt$ 
    - i. IF ( $Times4D(IT) > 0.0$ ) THEN
      1.  $FDFileNo = IT - 1$
      2. EXIT LOOP
    - c. END\_LOOP
14.  $FDTIME(Ind4Dold) = Times4D(FDFileNo)$ ; Set the time for this file.
15. Read the first set of files: IF ADVECT THEN Call *Load4DDData(Ind4Dold)* ELSE Call *LoadLESData( UnWind, FDFileNo, Ind4Dold, ErrStat )*; (***FDWind***)
16. Read the second set of files:
  - a.  $FDFileNo= FDFileNo+1$  (***FDWind***)
  - b. IF ADVECT THEN
    - i.  $FDFileNo = MOD(FDFileNo-1, Num4Dt) + 1$
    - ii. IF  $FDFileNo == 1$  THEN
      1.  $Shft4Dnew = Shft4Dnew + 1$  (***FDWind***)
      2. IF  $Ind4DAdv \leq NumAdvect$  (***FDWind***) THEN
        - a. IF  $MOD( Shft4Dnew, Num4Dx ) == 0$  THEN Call *ReadAll4DDData(UnWind, ErrStat)* (***FDWind***)
      - iii.  $FDTIME(Ind4Dnew) = Times4D(FDFileNo) + Shft4Dnew * FDPer$  (***FDWind***) ; Set the time for this file.
      - iv. Call *Load4DDData(Ind4Dnew)*
    - c. ELSE :
      - i.  $FDTIME(Ind4Dnew) = Times4D(FDFileNo)$  (parent module ***FDWind***) ; Set the time for this file.
      - ii. Call *LoadLESData( UnWind, FDFileNo, Ind4Dnew, ErrStat )*
  17. Set  $FDUnit = UnWind$  (***FDWind***)
  18. Set  $PrevTime= 0.0$  (***FDWind***)
  19. Set  $Initialized = .TRUE.$  (***FDWind***)

## OUTPUT

*Initialized:* (***FDWind***) Initialization flag

*ErrStat:* I; the usual error status flag.

It also sets several variables needed within ***FDWind***: *FDRecL*, *Num4DxD*, *Num4DyD*, *Num4DzD*, *Num4DxD1*, *Num4DyD1*, *Num4DzD1*, *Tm\_max*, *FDTimStp*, *FDTIME*, *FDper*, *TsclFact*, *T\_4D\_En*, *Xt*, *Yt*, *Zt*, *DelXgrid*, *DelYgrid*, *DelZgrid*, *Times4D*, *FD[u/v/w]*, *FD[u/v/w]Data*, *Ind4Dold*, *Ind4Dold*, *Shft4Dnew*, *PrevTime*, *FDUnit*, *FDFileNo*, *ADVECT*.

Other variables within ***FDWind*** are set through calls to subroutines.

## K2. READFDP(UNWIND, FILENAME, FDTSFILE, ERRSTAT)

Subroutine to read the input parameters for the LES. Called by *FDInit (FDWind)*.

### EXTERNAL (INVOKED) MODULES

From Parent Module *FDWind*[.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for reading wind files
<i>FileName</i>	C(*)	IN	Then name of the LES data file
<i>FDTSfile</i>	C(*)	OUT	The name of the file containing the time-step history of the wind files
<i>ErrStat</i>	I	OUT	If <>0, errors encountered

#### LOCAL VARIABLES

Name	Type	Value	Parameter/ Initialized	Description
<i>HeaderLine</i>	C(1024)			Header Line string
<i>Comp(3)</i>	C(1)	(/'U','V','W'/)PARAMETER		Wind components
<i>CoefTE</i>	R(ReKi)			Coefficient of thermal expansion
<i>DistScal</i>	R(ReKi)			Disturbance scale (ratio of wave height to rotor diameter) from input file
<i>Grav</i>	R(ReKi)			Gravitational acceleration
<i>LenScale</i>	R(ReKi)			Length scale (h)
<i>Ri</i>	R(ReKi)			Richardson number
<i>Ubot</i>	R(ReKi)			Steady u-component wind speed at the bottom of the wave
<i>Zm_maxo</i>	R(ReKi)			The non-dimensional vertical height of the untrimmed dataset
<i>Xm_max</i>	R(ReKi)			The nondimensional downwind length of the dataset
<i>Ym_max</i>	R(ReKi)			The nondimensional lateral width of the dataset
<i>Zm_max</i>	R(ReKi)			The nondimensional vertical height of the

		dataset
I	I	Counter

**STEPS**

1. Open the 4D parameter file: Call *OpenInpFile( UnWind, TRIM(BinFile), ErrStat )* (*NWTC\_IO* within *NWTC\_Library*), RETURN in case *ErrStat* is returned  $\neq 0$
2. Read the 4D wind parameters specific to this turbine simulation: Use *ReadStr*, *ReadCom* and *ReadVar* (*NWTC\_IO*), for strings, comments and actual variable values, and read the following:
  - a. *HeaderLine* (echo it on the screen for the user's sake)
  - b. *FDSPPath (FDWind)*; path to the binary dataset
  - c. *FDTStfile (FDWind)*; name of file with time-step history of the wind files
  - d. *Ubot*; Steady u-component of the wind at bottom of wave
  - e. *DistScal*; disturbance scale
  - f. *LX(FDWind)*; fractional location of tower centerline from upwind end of dataset
  - g. *LY(FDWind)*; fractional location of tower centerline from R to L (looking downwind) end of dataset
  - h. *LZ(FDWind)*; fractional location of hub height from bottom up of dataset
  - i. *T\_4D\_St (FDWind)*; start time of wave event
  - j. *ScaleVel (FDWind)*; Scaling velocity, U0: 0.5\* difference in wind speed between the top and bottom of the billow
  - k. *RotDiam (FDWind)*; Rotor diameter
  - l. *FD\_DF\_X (FDWind)*; decimation factor in X dir.
  - m. *FD\_DF\_Y (FDWind)*; decimation factor in Y dir.
  - n. *FD\_DF\_Z (FDWind)*; decimation factor in Z dir.
  - o. *VertShft (FDWind)*; flag to indicate whether or not to shift the Z values for the w component
  - p. *Xm\_max*; maximum non-dimensional downwind extent of dataset from its center
  - q. *Ym\_max*; maximum non-dimensional lateral extent of dataset from its center
  - r. *Zm\_max*; maximum non-dimensional vertical extent of dataset from its center
  - s. *Zm\_maxo*; maximum non-dimensional vertical extent of untrimmed dataset from its center
  - t. DO\_LOOP *I=1,3* read and set the following:
    - i. *Comp(I)*
    - ii. *ScalFact(I) (FDWind)*
    - iii. Set *ScalFact(I) = ScalFact(I) \* ScaleVel (FDWind)*
    - iv. *Offset(I) (FDWind)*
    - v. Set *Offsets(I)= Offsets(I) \* ScaleVel*
  - u. END\_LOOP
  - v. Set *Offsets (1) = Offsets (1) + ScaleVel + Ubot*; u-component offset to convert integer data to actual wind speeds

- w. Resume reading:
- x.  $Num4Dt$  (***FDWind***); number of LES grids, one per time step
- y.  $Num4Dx$  (***FDWind***); number of LES grids along x
- z.  $Num4Dy$  (***FDWind***); number of LES grids along y
- aa.  $Num4Dz$  (***FDWind***); number of LES grids along z
- bb.  $Ri$ ; Richardson number
- cc.  $CoefTe$ ; thermal expansion coefficient
- dd.  $Grav$ ; gravity
- ee. ***ADVECT*** (***FDWind***); Advection Flag
  - i. IF  $Errstat < 0$  THEN
    - 1. Set ***ADVECT***= .FALSE.
    - 2.  $Ind4DAdv=0$  (***FDWind***)
    - 3.  $ErrStat=0$
    - 4. Warn user of no advection to be used
  - ii. ELSE:
    - 1. IF ***ADVECT*** THEN
      - a. IF  $FD\_DF\_X \neq 1$  THEN Set  $FD\_DF\_X = 1$  (it has to be this way for ***ADVECT***, and warn user)
      - b. Read ***NumAdvect*** (***FDWind***) from ***UnWind***; number of 4D files
      - c. Check  $NumAdvect \geq 1$  else warn user and RETURN with  $ErrStat=1$
      - d. ALLOCATE IF .NOT. Allocated ***AdvFiles(NumAdvect)*** (***FDWind***)
      - e. Call **ReadAryLines** (***NWTC\_IO***), to read ***AdvFiles***, Advection File names
      - f. Set  $Ind4DAdv=1$
    - 2. ELSE Set  $Ind4DAdv=0$
- 3. Close ***UnWind*** file unit
- 4. Set some other variables:
  - a.  $LenScale = RotDiam * DistScal / Zm\_max$ ; Length scale (h).
  - b.  $Xmax$  (***FDWind***) =  $Xm\_max * LenScale$ ; The dimensional length of the dataset.
  - c.  $Ymax$  (***FDWind***) =  $Ym\_max * LenScale$ ; The dimensional width of the dataset
  - d.  $Zmax$  (***FDWind***) =  $Zm\_max * LenScale$ ; The dimensional vertical height of the dataset.
  - e.  $TSclFact$  (***FDWind***) =  $LenScale / ScaleVel$ ; Scale factor for time (h/U0).

## OUTPUT

It reads the LES parameter file and sets several variables belonging to the module ***FDWind***: ***FDSPPath***, ***FDTsfile***, ***Lx***, ***Ly***, ***Lz***, ***T\_4D\_St***, ***ScaleVel***, ***RotDiam***, ***FD\_DF\_X***, ***FD\_DF\_Y***, ***FD\_DF\_Z***, ***VertShft***, ***ScalFact***, ***Ind4DAdv***, ***ADVECT***, ***Num4Dt***, ***Num4Dx***, ***Num4Dy***, ***Num4Dz***, ***NumAdvect***, ***AdvFiles***, ***Xmax***, ***Ymax***, ***Zmax***, ***TSclFact***.

***ErrStat : I***; the usual error status flag.

### K3. READ4DTIMES (UNWIND, FILENAME, ERRSTAT)

This subroutine is used to read the time array for the 4D data. The times in the file are non-dimensional and non-uniformly spaced. They are scaled using  $TScIFact$  to obtain units of seconds and  $T_4D_St$  is added to allow the bilow to start at non-zero time. Called by *FDInit* (*FDWind*).

#### EXTERNAL (INVOKED) MODULES

From Parent Module *FDWind*[.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for reading wind files
<i>FileName</i>	C(*)	IN	Name of the LES data file
<i>ErrStat</i>	I	OUT	If $<>0$ , errors encountered

Additional Input comes from variables in parent module (e.g.: *Num4Dt* (*FDWind*))

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>I</i>	I			Counter

#### STEPS

1. ALLOCATE IF .NOT. Allocated:
  - a. *Times4D(Num4Dt)* (*FDWind*); 4D time array
  - b. *Times4DIx(Num4Dt)* (*FDWind*)
2. Open the 4D time file: Call *OpenFInpFile* (*UnWind*, TRIM(*FileName*), *ErrStat*)*(NWTC\_IO* within *NWTC\_Library* package)
3. Read and skip first line of comment
4. DO *I*=1,*Num4Dt*: Read from *UnWind* the times: *Times4DIx(I)*, *Times4D(I)* (in case *ErrStat* $<>0$  warn user of error in reading time-step file and RETURN ); END\_LOOP
5. Close *UnWind* file unit

#### OUTPUT

*ErrStat* : I; the usual error status flag.

It reads the 4D time-step file and sets some variables belonging to the module *FDWind*: *Times4D*, *Times4DIx*.

## K4. READALL4DDATA (UNWIND, ERRSTAT)

Subroutine to read the data into one array to be accessed later when ADVECT=.TRUE. Since there are just a few time steps, loading them into memory will (hopefully) save I/O time. Called by *FDInit (FDWind)*.

### EXTERNAL (INVOKED) MODULES

From Parent Module *FDWind*[.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for the file to open
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Times4Dlx (FDWind)*)

### LOCAL VARIABLES

Name	Type	Value	Parameter/ Initialized	Description
<i>IT</i>	I			Timestep counter
<i>FDNum</i>	C(1)			Index of time file
<i>DNSFileName</i>	C(20)			String containing part of the current file name

### STEPS

1. DO  $IT=1, Num4Dt$  ;through the timesteps
  - a. Write *Times4Dlx(IT) (FDWind)* into *FDNum* ; index of time file
  - b. Set *DNSFileName* = TRIM(*AdvFiles(Ind4DAdv)*)//'\_'//TRIM(*FDNum*)//'.dns' (*FDWind*)
  - c. Multiple Calls to *Read4DData (FDWind)* to get *FD[u/v/w]Data*:
    - i. Call *Read4DData( UnWind, TRIM(FDSpath )//'\u\w\w\w\w\_16i\_//TRIM(DNSFileName), FD[u/v/w]Data, IT, ScalFact([1/2/3]), Offsets([1/2/3]), ErrStat )*
2. END\_LOOP
3. Set *Ind4DAdv* = *Ind4DAdv* + 1 ; (*FDWind*)

### OUTPUT

*ErrStat*: I; the usual error flag

It sets other variables declared in parent module (*FDWind*): *FD[u/v/w]Data*, *Ind4DAdv*.

## K5. LOADESDATA (UNWIND, FILENO, INDEX, ERRSTAT)

Subroutine to read the binary data from the U, V, and W files; it stores them in the arrays  $FD_u$ ,  $FD_v$ , and  $FD_w$  (by calling *Read4DDData* (*FDWind*)). Called by *FDInit* (*FDWind*).

### EXTERNAL (INVOKED) MODULES

From Parent Module *FDWind*[.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for the wind file to open
<i>FileNo</i>	I	IN	Current file number to read
<i>Indx</i>	I	IN	Index into the data arrays (time index)
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *FDSpath* (*FDWind*))

### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>FDNum</i>	C(5)			Dummy 4-byte real number
<i>LESFileName</i>	C(20)			String containing part of the current file name

### STEPS

1. Write *FileNo* into *FDNum*; index of time file
2. Set *LESFileName* = TRIM(*FDNum*)//'.les'
3. Multiple Calls to *Read4DDData* (*FDWind*) to get *FD[u/v/w]*:
  - i. Call *Read4DDData*( *UnWind*, TRIM( *FDSpath* )//'[u\ v\ w]\ [u\ v\ w]\_16i\_//TRIM(*DNSFileName*), *FD[u/v/w]*, *Indx*, *ScalFact*([1/2/3]), *Offsets*([1/2/3]), *ErrStat* )( *FDWind* )

### OUTPUT

*ErrStat*: I; the usual error flag

It reads and sets variables belonging to the module *FDWind*: *FD[u/v/w]*.

## K6. READ4DDATA(UNWIND, FILENAME, COMP, IDX4, SCALE, OFFSET, ERRSTAT)

Subroutine to read 1 time-step's worth of large-eddy wind data for one component from a file. Called by *ReadAll4DData*, *LoadLESData* (*FDWind*).

### EXTERNAL (INVOKED) MODULES

From Parent Module *FDWind*[.f90]

#### INPUT

Name	Type	INTENT	Description
<i>UnWind</i>	I	IN	unit number for the wind file to open
<i>FileName</i>	C(*)	IN	Name of the LES data file
<i>Comp(:,:,:,:)</i>	R(ReKi)	INOUT	( <i>Num4Dx</i> , <i>Num4Dy</i> , <i>Num4Dz</i> , <i>Num4Dt</i> ) The velocity array ( <i>FDWind</i> )
<i>Idx4</i>	I	IN	The index of the 4th dimension of Comp, which is to be read
<i>Scale</i>	R(ReKi)	IN	Scale factor for converting from integers to non-normalized reals
<i>Offset</i>	R(ReKi)	IN	The offset for converting from integers to non-normalized reals
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Num4Dy* (*FDWind*))

#### LOCAL VARIABLES

Name	Type	Value	Parameter/ Initialized	Description
<i>IX</i>	I			Loop counter for x dir
<i>IXK</i>	I			Loop counter for decimated x dir
<i>IY</i>	I			Loop counter for y dir
<i>IYK</i>	I			Loop counter for decimated y dir
<i>IZ</i>	I			Loop counter for z dir
<i>IZK</i>	I			Loop counter for decimated z dir
<i>Com</i> ( <i>Num4Dx</i> , <i>Num4Dy</i> )	I(B2Ki)			Temporary array to hold component's integer values for a given Z

## STEPS

1. Open binary file: Call *OpenUIInBEFile* ( *UnWind*, TRIM(*WindFile*), *ErrStat*)\_(*NWTC\_IO* within *NWTC\_Library* package) (big-endian format data expected)
2. Initialize : *IZK*=0
3. Read data: DO\_LOOP *IZ*=1,*Num4Dz*, *FD\_DF\_Z* (*FDWind*)
  - a. Read *Com* from *UnWind* (IF *ErrStat* <>0 user is warned and RETURN)
  - b. Set *IZK*=*IZK*+1
  - c. Set *IYK*=0
  - d. DO\_LOOP *IY*=1,*Num4Dy*, *FD\_DF\_Y* (*FDWind*):
    - i. Set *IYK*=*IYK*+1
    - ii. DO\_LOOP *IX*=1,*Num4Dx*, *FD\_DF\_X* (*FDWind*):
      1. *IXK* = ( MOD(*IX*+*Shft4Dnew*-1,*Num4Dx*) + *FD\_DF\_X* )/*FD\_DF\_X* (*FDWind*): shift the x-index to perform advection
      2. *Comp*(*IXK*,*IYK*,*IZK*,*Indx4*) = *Scale*\**Com*(*IX*,*IY*) + *Offset* (*FDWind*): the actual velocity
    - iii. END\_LOOP
  - e. END\_LOOP
4. END\_LOOP
5. Close(*Unwind*), i.e. the *FileName* file

## OUTPUT

*ErrStat*: *I*; the usual error flag

*Comp* (*Num4Dx*, *Num4Dy*, *Num4Dz*, *Num4Dt*): *R(ReKi)*: It reads the 4D data file (that contains either u, or v, or w components) and extracts velocities and scales them.

## K7. LOAD4DDATA(INPINDX)

Subroutine that takes the data from the storage array (used when ADVECT=.TRUE.), shifts it if necessary, and loads it into the main array for the time slice indexed by *InpIndx*. Called by *FDInit* (*FDWind*).

### EXTERNAL (INVOKED) MODULES

From Parent Module *FDWind*[.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>InpIndx</i>	I	IN	Index of time slice

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Num4Dx* (*FDWind*))

### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>IX</i>	I			Loop counter for x dir
<i>IXK</i>	I			Loop counter for decimated x dir

### STEPS

1. Read data from *FD[u/v/w]Data*: DO\_LOOP *IX*=1,*Num4Dx*, *FD\_DF\_X* (*FDWind*):
  - a. *IXK* = ( MOD(*IX*+*Shft4Dnew*-1,*Num4Dx*) + *FD\_DF\_X* )/*FD\_DF\_X* (*FDWind*): shift the x-index to perform advection
  - b. *FD[u/v/w](IXK,:,:InpIndx)* = *FD[u/v/w]Data (IX,:,:FDFileNo)* (*FDWind*); the actual velocity
2. END\_LOOP

### OUTPUT

*ErrStat*: I; the usual error flag

It reads the 4D array *FD[u/v/w]Data* (*FDWind*) (that contains either u, or v, or w components) and extracts *FD[u/v/w]* (*FDWind*) accounting for advection.

## K8. FD\_GETWINDSPEED (TIME, INPUTPOSITION, ERRSTAT)

This function is used to interpolate into the 4D wind arrays. It receives X, Y, Z and TIME from the calling routine. Called by WindInf\_getVelocity (InflowWind).

### THEORY

The time since the start of the 4D data is used to decide which pair of time slices to interpolate within and between. After finding the two time slices, it decides which eight grid points bound the (X,Y,Z) pair. It does a trilinear interpolation for each time slice. Linear interpolation is then used to interpolate between time slices. This routine assumes that X is downwind, Y is to the left when looking downwind and Z is up. It also assumes that no extrapolation will be needed except in time and the Z direction. In those cases, the appropriate steady winds are used. An important assumption is that the simulation time step is smaller than the wind-file time step.

### EXTERNAL (INVOKED) MODULES

From Parent Module *FDWind*[.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>Time</i>	R(ReKi)	IN	Time
<i>InputPosition</i> (3)	R(ReKi)	IN	Position (X, Y, Z)
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Initialized (FDWind)*)

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>Ixhyz</i>	R(ReKi)			Temporary interpolated value.
<i>Ixlyz</i>	R(ReKi)			Temporary interpolated value.
<i>Ixyz0</i>	R(ReKi)			Temporary interpolated value.
<i>Iyhz</i>	R(ReKi)			Temporary interpolated value.
<i>Iylz</i>	R(ReKi)			Temporary interpolated value.
<i>Ixyzn</i>	R(ReKi)			Temporary interpolated value.
<i>Tgrid</i>	R(ReKi)			Fractional distance between time grids.
<i>Xgrid</i>	R(ReKi)			Fractional distance between grids in the x direction.
<i>Xnorm</i>	R(ReKi)			Nondimensional downwind distance of the

		analysis point from upwind end of dataset.
$Ygrid$	R(ReKi)	Fractional distance between grids in the y direction.
$Ynorm$	R(ReKi)	Nondimensional lateral distance of the analysis point from right side of dataset (looking downwind).
$Zgrid$	R(ReKi)	Fractional distance between grids in the z direction.
$Zgrid_w$	R(ReKi)	Fractional distance between grids in the z direction for the w component.
$Znorm$	R(ReKi)	Nondimensional vertical distance of the analysis point from bottom of dataset.
$Znorm_w$	R(ReKi)	Nondimensional vertical distance of the analysis point from bottom of dataset for the w component.
$IT$	I	Counter
$IXHI$	I	Index for higher x
$IXLO$	I	Index for lower x
$IYHI$	I	Index for higher y
$IYLO$	I	Index for lower y
$IZHI$	I	Index for higher z
$IZHI_w$	I	Index for higher z for w-component
$IZLO$	I	Index for lower z
$IZLO_w$	I	Index for lower z for w-component

### STEPS

1. Make sure that ***FDWind*** module is initialized:
  - a. IF .NOT. *Initialized (FDWind)* THEN:
    - i. warn user on the screen
    - ii. Set *ErrStat*=1
    - iii. RETURN
  - b. ELSE Set *ErrStat*=0
2. If the TIME is greater than the time for the last file read, read another set of files until we straddle the current time. Stick to the last file if we have exhausted the data. Assumption is that the simulation time step is smaller than the wind-file time step:
3. IF *Time < PrevTime* .AND. *Time < FDTime(Ind4Dold)* (***FDWind***) THEN: going backwards

- a. Set  $Ind4Dold = 1$  and  $Ind4Dnew = 2$  (**FDWind**)
- b.  $FDFileNo$  (**FDWind**) =  $Num4Dt$  (**FDWind**)
- c. DO\_LOOP  $IT=1, Num4Dt$ :
  - i. IF  $Times4D(IT)$  (**FDWind**) >  $Time$  THEN
    - 1.  $FDFileNo = IT - 1$
    - 2. EXIT LOOP
  - d. END\_LOOP
  - e.  $FDTIME(Ind4Dold) = Times4D(FDFileNo)$  (**FDWind**); set the time for this file
  - f. Read 1<sup>st</sup> set of files:
    - i. IF ADVECT THEN Call **Load4DDData**( $Ind4Dold$ ) (**FDWind**); load data stored in  $FD[u/v/w]Data$
    - ii. ELSE Call **LoadLESData**( $FDUnit, FDFileNo, Ind4Dold, ErrStat$ ) (**FDWind**);
  - g. Read 2<sup>nd</sup> set of files:
    - i.  $FDFileNo = \text{MIN}(FDFileNo-1, Num4Dt)$
    - ii.  $Shft4Dnew = 0$ ; (**FDWind**)
    - iii. IF ADVECT THEN
      - 1.  $FDFileNo = \text{MOD}(FDFileNo-1, Num4Dt) + 1$
      - 2. IF  $FDFileNo == 1$  THEN
        - a.  $Shft4Dnew = Shft4Dnew + 1$
        - b. IF  $Ind4DAdv \leq NumAdvect$  (**FDWind**) THEN
          - i. IF  $\text{MOD}(Shft4Dnew, Num4Dx)$  (**FDWind**) == 0 THEN Call **ReadAll4DDData**( $FDUnit, ErrStat$ ) (**FDWind**)
        - 3.  $FDTIME(Ind4Dnew) = Times4D(FDFileNo) + Shft4Dnew * FDPer$  (**FDWind**); Set the time for this file.
        - 4. Call **Load4DDData**( $Ind4Dnew$ ); shift the data
      - iv. ELSE : no advection:
        - 1.  $FDTIME(Ind4Dnew) = Times4D(FDFileNo)$ ; Set the time for this file.
        - 2. Call **LoadLESData**( $FDUnit, FDFileNo, Ind4Dnew, ErrStat$ )
  - 4. WHILE\_LOOP  $Time > FDTIME(Ind4Dnew)$  .AND. ( $Time < T\_4D\_En(FDWind)$  .OR.  $ADVECT(FDWind)$ )
    - a.  $Ind4Dnew = Ind4Dold$  (reverse array indices)
    - b.  $Ind4Dold = 3 - Ind4Dnew$
    - c.  $FDFileNo = FDFileNo + 1$ ; increment file number
    - d. IF ADVECT THEN
      - i.  $FDFileNo = \text{MOD}(FDFileNo-1, Num4Dt) + 1$
      - ii. IF  $FDFileNo == 1$  THEN
        - 1.  $Shft4Dnew = Shft4Dnew + 1$
        - 2. IF  $Ind4DAdv \leq NumAdvect$  (**FDWind**) THEN
          - a. IF  $\text{MOD}(Shft4Dnew, Num4Dx) == 0$  THEN Call **ReadAll4DDData**( $FDUnit, ErrStat$ )

- iii.  $FDTIME(Ind4Dnew) = Times4D(FDFileNo) + Shft4Dnew * FDPer$  (**FDWind**); Set the time for this file.
- iv. Call **Load4DData**(*Ind4Dnew*) ; shift the data
- e. ELSE: no advection:
  - i.  $FDTIME(Ind4Dnew) = Times4D(FDFileNo)$  ; Set the time for this file.
  - ii. Call **LoadLESData**( *FDUnit*, *FDFileNo*, *Ind4Dnew*, *ErrStat* )
- 5. END\_LOOP
- 6. Find how far we are between grids in time and between grid points in each direction to interpolate later:
- 7.  $Tgrid = \text{MIN}(\text{MAX}((Time - FDTIME(Ind4Dold)) / (FDTIME(Ind4Dnew) - FDTIME(Ind4Dold)), 0.0), 1.0)$
- 8. Get values for interpolation in X (note grid is periodic in X):
  - a.  $Xnorm = (Xt + InputPosition(1)) / Xmax$  (**FDWind**)
  - b. WHILE\_LOOP  $Xnorm < 0.0$ :  $Xnorm = Xnorm + 1.0$  END\_LOOP: ensure  $Xnorm > 0$
  - c.  $Xgrid = \text{MIN}(\text{MAX}(\text{MOD}(Xnorm, DelXgrid), 0.0), 1.0)$
  - d.  $IXLo = \text{MAX}(\text{MOD}(\text{INT}(Xnorm * Num4DxD1), 1), Num4DxD1)$ , 1 )
  - e.  $IXHi = \text{MOD}(IXLo, Num4DxD) + 1$
- 9. Get values for interpolation in Y (note grid is periodic in Y):
  - a.  $Ynorm = (Yt + InputPosition(2)) / Ymax$
  - b. WHILE\_LOOP  $Ynorm < 0.0$ :  $Ynorm = Ynorm + 1.0$  END\_LOOP: ensure  $Ynorm > 0$
  - c.  $Ygrid = \text{MIN}(\text{MAX}(\text{MOD}(Ynorm, Delygrid), 0.0), 1.0)$
  - d.  $IYLo = \text{MAX}(\text{MOD}(\text{INT}(Ynorm * Num4DyD1), 1), Num4DyD1)$ , 1 )
  - e.  $IYHi = \text{MOD}(IYLo, Num4DyD) + 1$
- 10. Get values for interpolation in Z: linear interpolation;nearest-neighbr extrapolation:
  - a.  $Znorm = \text{MIN}(\text{MAX}((Zt + InputPosition(3)) - ZRef(FDWind)) / Zmax, 0.0), 1.0$
  - b.  $Zgrid = \text{MIN}(\text{MOD}(Znorm, DelZgrid), 0.0), 1.0$
  - c.  $IZLo = \text{MAX}(\text{INT}(Znorm * Num4DzD1), 1)$
  - d. IF (  $IZLo == Num4DzD$  (**FDWind**) ) THEN: at the upper end of z dimension, decrement index and set grid coordinate to 1:
    - i.  $IZLo = Num4DzD1$
    - ii.  $Zgrid = 1.0$
  - e. Set  $IZHi = IZLo + 1$
  - f. Find the equivalent Znorm (Znorm\_w) for the w-component, which may be shifted vertically by half the original grid spacing:
    - i. IF  $VertShft$  (**FDWind**) THEN  $Znorm_w = \text{MAX}(Znorm - 0.5 * DelZgrid / FD_DF_Z, 0.0)$
    - ii. ELSE  $Znorm_w = Znorm$
    - iii.  $Zgrid_w = \text{MIN}(\text{MAX}(\text{MOD}(Znorm_w, DelZgrid), 0.0), 1.0)$
    - iv.  $IZLo_w = \text{MAX}(\text{INT}(Znorm_w * Num4DzD1), 1)$

- v. IF ( $IZLo_w == Num4DzD (FDWind)$ ) THEN: at the upper end of z dimension, decrement index and set grid coordinate to 1:
  - vi.  $IZLo_w = Num4DzD1$
  - vii.  $Zgrid_w = 1.0$
  - viii. Set  $IZHi_w = IZLo_w + 1$
11. Interpolate for u FD[u/v/w] component of wind within the grid:
- a.  $Iylz = ( FD[u/v/w] (IXLo,IYLo,IZHi,Ind4Dold) - FD[u/v/w] (IXLo,IYLo,IZLo,Ind4Dold) ) * Zgrid + FD[u/v/w] (IXLo,IYLo,IZLo,Ind4Dold)$ ; interpolate along z, with low x and low y
  - b.  $Iyhz = ( FD[u/v/w] (IXLo,IYHi,IZHi,Ind4Dold) - FD[u/v/w] (IXLo,IYHi,IZLo,Ind4Dold) ) * Zgrid + FD[u/v/w] (IXLo,IYHi,IZLo,Ind4Dold)$ ; interpolate along z, with low x and high y
  - c.  $Ixlyz = ( Iyhz - Iylz ) * Ygrid + Iylz$ ; interpolate along y between 2 above values
  - d. Repeat steps a-c above with  $IXHi$  in place of  $IXLo$ , i.e. at high x, and get  $Ixhyz$
  - e.  $Ixyz0 = ( Ixhyz - Ixlyz ) * Xgrid + Ixlyz$ ; interpolate along x the above values  $Ixlyz$  and  $Ixhyz$
  - f. Repeat steps a-e above with  $Ind4Dnew$  in place of  $Ind4Dold$ , i.e. at high time step, and get  $Ixyzn$  as final value
  - g.  $FD\_GetWindSpeed\%Velocity([1/2/3]) = ( Ixyzn - Ixyz0 ) * Tgrid + Ixyz0$ ; final time interpolation

12. Set  $PrevTime (FDWind) = Time$

## OUTPUT

*ErrStat: I*; the usual error flag.

It computes and stores  $FD\_GetWindSpeed$ : TYPE(**InflIntrpOut**) (*SharedInflowDefns*); the wind velocity (no tower effect) at the point of interest following the FD file type.

It also sets  $PrevTime (FDWind)$ .

## K9. FD\_GETRVALUE(RVARNOME, ERRSTAT)

Function that returns a real scalar value whose name is listed in the VarName input argument and that belongs to the module FDWind. If the name is not recognized, an error is returned in ErrStat. Called by WindInf, ADHack, DiskVel and (InflowWind).

### EXTERNAL (INVOKED) MODULES

From Parent Module *FDWind*[.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>RVarName</i>	C(*)	IN	Variable name string
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Initialized (FDWind)*)

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>VarNameUC</i>	C(20)			Upper case <i>RVarName</i>

#### STEPS

1. IF .NOT. *Initialized* (parent module *FDWind*): module NOT initialized: THEN:
  - a. warn user and RETURN with *ErrStat*=1
2. ELSE initialize *ErrStat*=0
3. Transform to upper case *VarNameUC* = *RVarName*
4. Based on *VarNameUC*:
  - a. 'ROTDIAM' → *FD\_GetValue* = *RotDiam* (*FDWind*)
  - b. All other cases: warn user of invalid variable name and set *ErrStat*=1

#### OUTPUT

*FD\_GetValue*: R(ReKi); the value of the variable whose name string was input.

*ErrStat*: I; the usual error flag.

## K10. FD\_TERMINATE (ERRSTAT)

Subroutine to close files, deallocate memory, and un-set the initialization flag. Called by *WindInf Terminate (InflowWind)*.

### EXTERNAL (INVOKED) MODULES

From Parent Module *FDWind*[.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Initialized (FDWind)*)

### LOCAL VARIABLES

N/A

### STEPS

1. Close *FDunit* file
2. Set *ErrStat*=0
3. DEALLOCATE if allocated :
  - a. *FDu, FDv, FDw, FDuData, FDvData , FDwData, Times4D, Times4DIx, AdvFiles* (*FDWind*)
4. Set *Initialized* = .FALSE.; initialization flag: if T=initialized

### OUTPUT

*ErrStat*: I; the usual error flag

It deallocates memory and closes file.

## APPENDIX L. CTWIND MODULE

### GENERAL ORGANIZATION

This module uses/reads coherent turbulence parameter (CTP) files and processes the data in them to get coherent turbulence, which is later superimposed on a background wind field (the super-positioning occurs elsewhere). The turbulence in this module is part of the KH billow, which can be read using *FDWind*. As a result, the scaling here should be similar to *FDWind*.

This module assumes that the origin, (0,0,0), is located at the tower centerline at ground level, and that all units are specified in the metric system (using meters and seconds). Data is shifted by half the grid width when used with *FFWind*.

There are 3 files read:

1. A .ctp file containing basic information on name of background file and .cts file
2. A .cts file containing the extension for the turbulence file, and some scaling factors, and the time steps for the turbulent events
3. A scale file, containing scaling factors and number of grid points in y and z direction

### EXTERNAL MODULES

- *NWTC\_Library* [see NWTC\_Library.f90]
- *SharedInflowDefns* [see SharedInflowDefs.f90]

### PRIVATE (LOCALLY DECLARED) DATA

Name	Type	Value	Parameter/ Initialized	Description
<i>numComps</i>	I	3	PARAMETER	Number of Components
<i>DelYCTgrid</i>	R(ReKi)			The nondimensional distance between grid points in the y direction.
<i>DelZCTgrid</i>	R(ReKi)			The nondimensional distance between grid points in the z direction.
<i>CTDistSc</i>	R(ReKi)			Disturbance scale (ratio of wave height to rotor diameter).
<i>CTOffset(NumComps)</i>	R(ReKi)			Offsets to convert integer data to actual wind speeds.
<i>CTSscale (NumComps)</i>	R(ReKi)			Scaling factors to convert integer data to actual wind speeds.
<i>CTvelU</i> (:,:,:) R(ReKi)				The y-z grid velocity data (U components) for the lower- and upper-bound time slices

<i>CTvelV</i> (:,:,:)	R(ReKi)	The y-z grid velocity data (V components) for the lower- and upper-bound time slices
<i>CTvelW</i> (:,:,:)	R(ReKi)	The y-z grid velocity data (W components) for the lower- and upper-bound time slices
<i>CTLy</i>	R(ReKi)	Fractional location of tower centerline from right (looking downwind) to left side of the dataset.
<i>CTLz</i>	R(ReKi)	Fractional location of hub height from bottom to top of dataset.
<i>CTSscaleVel</i>	R(ReKi)	Scaling velocity, U0. 2*U0 is the difference in wind speed between the top and bottom of the wave.
<i>CTYt</i>	R(ReKi)	Distance of the tower from the right side of the dataset (looking downwind).
<i>CT_Zref</i>	R(ReKi)	The reference height for the CT file (the bottom of the billow)
<i>CTYHWid</i>	R(ReKi)	The half the width of the background dataset, used to compute the CTwind time offset
<i>CTYmax</i>	R(ReKi)	The dimensional lateral width of the dataset.
<i>CTZmax</i>	R(ReKi)	The dimensional vertical height of the dataset.
<i>InvMCTWS</i>	R(ReKi)	The multiplicative inverse of the mean hub height wind speed for the CT wind data
<i>CT_DF_Y</i>	I	The decimation factor for the CT wind data in the y direction.
<i>CT_DF_Z</i>	I	The decimation factor for the CT wind data in the z direction.
<i>CTvel_files(2)</i>	I	Times for the CT wind files stored in CTvel arrays.
<i>IndCT_hi</i>	I	An index into the 3rd dimension of the

				CTvel arrays, indicating the upper time slice (allows us to avoid copying array)
<i>IndCT_lo</i>	I			An index into the 3rd dimension of the CTvel arrays, indicating the lower time slice (allows us to avoid copying array)
<i>NumCTt</i>	I			The number of CT wind grids, no more than one grid per time step.
<i>NumCTy</i>	I			The number of CT wind grid points in the y direction.
<i>NumCTyD</i>	I			The decimated number of CT wind grid points in the y direction.
<i>NumCTyD1</i>	I			The decimated number of CT wind grid points in the y direction minus 1.
<i>NumCTz</i>	I			The number of CT wind grid points in the z direction.
<i>NumCTzD</i>	I			The decimated number of CT wind grid points in the z direction.
<i>NumCTzD1</i>	I			The decimated number of CT wind grid points in the z direction minus 1.
<i>TimeIndx</i>	I	0	Initialized + SAVE	Initialization Flag; if =1 it is initialized
<i>Tdata</i> (:)	R(ReKi)			The list of times for the CT-wind input files.
<i>TimeStpCT</i> (:)	I			The list of time steps from the original LES simulation, associated with the CT-wind times.
<i>CTWindUnit</i>	I			unit number used to read the wind files at each call to <u><i>CT_GetWindSpeed()</i></u>
<i>CTVertShft</i>	L			Flag to indicate whether or not to shift the z values for the w component.
<i>CText</i>	C(3)			The extension used for coherent turbulence data files. (usually "les" or "dns")

<i>CTSpPath</i>	C(1024)	The path to the CT wind files.
-----------------	---------	--------------------------------

- **CTWindFiles:** NEW PRIVATE TYPE made up of 2 string fields:

Name	Type	Value	Parameter/ Initialized	Description
<i>CTTSfile</i>	C(1024)			The name of the file containing the time-step history for the coherent turbulence events.
<i>CTbackgr</i>	C(1024)			The name of the background wind data

- New PRIVATE Routines and Functions

Name	Description
<u><i>ReadCTData</i></u>	
<u><i>LoadCTData</i></u>	
<u><i>ReadCTP</i></u>	Reads the CTP file
<u><i>ReadCTTS</i></u>	Reads the CTTS file
<u><i>ReadCTScales</i></u>	

#### PUBLIC (LOCALLY DECLARED) DATA

- **CT\_Backgr:** NEW PUBLIC TYPE made up of 1 string, 1 Integer, and 1 logical fields:

Name	Type	Value	Parameter/ Initialized	Description
<i>WindFile</i>	C(1024)			The name of the background wind file
<i>WindFileType</i>	I			The type of background wind file (currently only FF)
<i>CoherentStr</i>	L			If the coherent time step file is blank or doesn't exist, this is FALSE (use the background only)

- New PUBLIC Routines and Functions

Name	Description
<u><i>CT_Init</i></u>	Initialization routine
<u><i>CT_SetRefVal</i></u>	Sets reference values for grid shift and reference height
<u><i>CT_GetWindSpeed</i></u>	
<u><i>CT_Terminate</i></u>	

## L1. CTINIT (UNWIND, WINDFILE, BACKGRNDVALUES, ERRSTAT)

Subroutine called at the beginning of a simulation. It reads the CTP file to obtain the name of the CTS file, the path locating the binary KH files, and decimation factors.

It returns the background wind file and type; it also returns a flag that determines if CT wind files are ACTUALLY to be used (e.g., if the CTS file is blank or there is one line of zero in the CTS time array).

Called by WindInf Init (InflowWind).

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for reading wind files
<i>WindFile</i>	C(*)	IN	Name of the CTP (.ctp) wind file
<i>BackGrndValues</i>	CT_Backgr (CTWind)	OUT	output background values
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *FF\_Wind (SharedInflowDefns)*)

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>CTP_files</i>	CTWindFiles (CTWind)			See <b>CTWindFiles</b>
<i>CT_SC_ext</i>	C(*)			Extension of the scaling file

#### STEPS

1. IF *TimeIndx* (parent module *CTWind*) $\neq 0$ : module already initialized: THEN:
  - a. warn user and RETURN with *ErrStat*=1
2. ELSE *ErrStat*=0 and Call NWTC Init (NWTC\_Library)
3. Read input CTP File: Call ReadCTP( UnWind, WindFile, CTP\_files, ErrStat ) (CTWind), RETURN in case *ErrStat* is returned  $\neq 0$
4. Set output:
  - a. *BackGrndValues%WindFileType* = *CTP\_files%CTbackgr* : filename for the background file
  - b. *BackGrndValues%WindFileType* = *FF\_Wind (SharedInflowDefns)* :The background file type is currently set as **FF type only**, i.e. full field type
5. Call ReadCTTS( UnWind, CTP\_files%CTTSfile, CT\_SC\_ext, ErrStat ) (CTWind),to read CTTS file and get time steps and file number arrays and check returned *ErrStat*:

- a. IF (*ErrStat* == 0 .AND. *NumCTt* (*CTWind* and initialized by *ReadCTTS*),> 1) THEN:
    - i. Set *BackGrndValues%CoherentStr* = .TRUE.
    - ii. Call *ReadCTScales*( *UnWind*, TRIM( *CTSpPath* )//'\Scales.'//TRIM( *CT\_SC\_ext* ), *ErrStat* ) (*CTWind*), i.e. Read file containing scaling for the binary large-eddy files, RETURN incase *ErrStat* is returned <>0
    - iii. Set *CTScale(:)* = *CTScaleVel*\**CTScale(:)* (variables calculated in the previous subroutine call and defined in the parent module *CTWind*)
    - iv. Set *CTOffset (:)* = *CTScaleVel*\* *CTOffset (:)* (variables calculated in the previous subroutine call and defined in the parent module *CTWind*)
  - b. ELSE:
    - i. IF *ErrStat* <=0 THEN :The file is missing, blank (or possibly incomplete), or has only 1 time step line (which is zero); Go on without the CT file, using just the background:
      - 1. Warn the user that Coherent Turbulence Wind File is turned off
      - 2. Set *BackGrndValues%CoherentStr* = .FALSE.
      - 3. Call *CT\_Terminate*( *ErrStat* ) (*CTWind*)
    - ii. RETURN
6. Set the following variables (these are actually constants from this point on, belonging to the parent module *CTWind*) based on other variables set in the previous two calls to reading routines
- a. *CTYHWid* = 0.0; This value is used to perform a time shift (the equivalent distance of *FFYHWid* [approx. rotor radius])
  - b. *CT\_Zref* = -1.0; This value needs to be set after the corresponding background turbulence has been read (or the CTS file should be changed)
  - c. *NumCTyD* = ( *NumCTy* + *CT\_DF\_Y* - 1 )/*CT\_DF\_Y* ; The decimated number of CT wind grid points in the y direction.
  - d. *NumCTzD* = ( *NumCTz* + *CT\_DF\_Z* - 1 )/*CT\_DF\_Z* ; The decimated number of CT wind grid points in the z direction.
  - e. *NumCTyD1* = *NumCTyD* - 1 ; The decimated number of CT wind grid points in the y direction minus 1.
  - f. *NumCTzD1* = *NumCTzD* - 1 ; The decimated number of CT wind grid points in the z direction minus 1.
  - g. *CTYt* = *CTYmax*\**CTLy* ; Distance of the tower from the right side of the dataset (looking downwind).
  - h. *CTZt* = *CTZmax*\**CTLz*; Distance of the hub from the bottom of the dataset.
  - i. *DelYCTgrid* = 1.0/*NumCTyD1*; The nondimensional distance between grid points in the y direction.
  - j. *DelZCTgrid* = 1.0/*NumCTzD1* ; The nondimensional distance between grid points in the z direction.
7. ALLOCATE some arrays if not ALLOCATED and initialize to 0.0:
- a. *CTvelU*(*NumCTyD*,*NumCTzD*,2) (the original velocity data) = 0.0

- b.  $CTvelV(NumCTyD, NumCTzD, 2)$  (the original velocity data) = 0.0
  - c.  $CTvelW(NumCTyD, NumCTzD, 2)$  (the original velocity data) = 0.0
8. Initialize more variables:
- a.  $CTvel\_files(:) = 0$  : the name of the files currently in the  $CTvel$  array
  - b.  $CTWindUnit = UnWind$  : This unit is needed to open the binary files at each step
  - c.  $TimeIndx = 1$  ; Initialization Flag ; if =1 it is initialized

## **OUTPUT**

*BackGrndValues: TYPE(CT\_Backgr) (CTWind)*

*ErrStat : I*

It also sets certain variables needed within CTWind:  $CTYHWid$ ,  $CT\_Zref$ ,  $NumCTyD$ ,  $NumCTzD$ ,  $NumCTyD1$ ,  $NumCTzD1$ ,  $CTYt$ ,  $CTZt$ ,  $DelYCTgrid$ ,  $DelZCTgrid$ ,  $CTvel[U/V/W](::,:)$ ,  $CTvel\_files(:)$ ,  $CTWindUnit$ ,  $TimeIndx$ .

## L2. READCTP (UNWIND, FILENAME, CTPSCALING, ERRSTAT)

Subroutine to read the input parameters for the coherent turbulence events, based on the large-eddy simulation (LES). Called by *CTInit* (*CTWind*).

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for reading wind files
<i>FileName</i>	C(*)	IN	Name of the input CTP (.ctp) wind file
<i>CTPscaling</i>	<i>CTWindFiles</i> ( <i>CTWind</i> )	OUT	File names contained in the CTP file
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *CTSPath* (*CTWind*)).

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>HeaderLine</i>	C(1024)			Header text in the .ctp file
<i>TmpPath</i>	C(1024)			Temporary path retrieved

#### STEPS

1. Open the CTP input file *FileName* Call *OpenFInpFile* (*UnWind*, TRIM( *FileName* ), *ErrStat*) (*NWTC\_IO* within *NWTC\_Library* package) and RETURN IF *ErrStat*  $\neq 0$
2. Read the CTP Input file using *ReadStr*, *ReadCom* and *ReadVar* (*NWTC\_IO* within *NWTC\_Library* package):
  - r. Read the header and sub-header
  - s. From the file Read the *CTSPath* (*CTWind*), and i.e. the path to the binary coherent turbulence dataset
  - t. From the file Read *CTPscaling%CTTSfile* (*CTWind*), the name of the “.cts” file containing the time steps for the coherent turbulence events; also if there is no path, make sure we can get a path (*TmpPath*) for it from *FileName* by using *GetPath* (*NWTC\_IO* within *NWTC\_Library* package)
  - u. From the file Read *CTPscaling%CTbackgr* (*CTWind*), name of file containing the background wind; also if there is no path, make sure we can get a path (*TmpPath*) for it from *FileName* by using *GetPath*

- v. Read the variables *CT\_DF\_Y* and *CT\_DF\_Z* decimation factors for wind data in the Y and Z directions.
3. Close(*Unwind*), i.e. the *FileName* file

## OUTPUT

*CT\_DF\_Y*, *CT\_DF\_Z* (*CTWind*): Integer: The decimation factor for the CT wind data in the y and z directions.

*CTSpaht* (*CTWind*): C(1024); The path to the CT wind files.

*CTPscaling%CTTSfile* (*CTWind*): C(1024); The name of the file containing the time-step history of the for the coherent turbulence events.

*CTPscaling%CTbackgr* (*CTWind*): C(1024); The name of the file containing the background wind data.

### L3. READCTTS (UNWIND, FILENAME, CT\_SC\_EXT, ERRSTAT)

Subroutine to read the input parameters calculated in TurbSim for the scaling of coherent turbulence events. It reads the .cts file and saves the time step and file number arrays. Called by *CTInit* (*CTWind*).

#### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for reading wind files
<i>FileName</i>	C(*)	IN	Name of the input CTTs wind file
<i>CT_SC_ext</i>	C(3)	OUT	The extension used for coherent turbulence scale files.(usually "les", "dns", or "dat")
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *NumCTt* (*CTWind*)).

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>IT</i>	I			Loop counter

#### STEPS

1. Initialize *NumCTt*= 0 (*CTWind*); The number of CT wind grids.
2. Open the CTS input file *FileName* Call *OpenFInpFile* (*UnWind*, TRIM( *FileName* ), *ErrStat*) (*NWTC\_IO* within *NWTC\_Library* package) and RETURN IF *ErrStat*  $\neq 0$
3. READ *CTScaleVel* (*CTWind*) as numeric value and REWIND the file: IF *ErrStat*  $\neq 0$  exiting from READ THEN the file type is “new”, and there is no numeric value at the beginning:
  - a. Use *ReadVar* (*NWTC\_IO* within *NWTC\_Library* package) and READ
    - i. *CText* : (*CTWind*); the extension for the turbulence file
    - ii. Assign *CT\_SC\_ext* = *CText*
    - iii. Read *CTScaleVel* scaling factor
4. ELSE (file type is “old”, and there is a numeric value at the beginning):
  - a. Read in *CTScaleVel* (*ReadVar*)
  - b. Set *CText* = ‘les’ and *CT\_SC\_ext* = ‘dat’
5. Use *ReadVar* and READ the following variables declared in the parent module *CTWind*
  - a. *InvMCTWS* :The inverse of the mean hub height wind speed for the CT wind data
  - b. *CTYmax* :The dimensional lateral width of the dataset.
  - c. *CTZmax* :The dimensional vertical height of the dataset.
  - d. *CTDistSc*: Disturbance scale (ratio of wave height to rotor diameter).

- e.  $CTLy$ : Fractional location of tower centerline from right (looking downwind) to left side of the dataset.
  - f.  $CTLz$ : Fractional location of hub height from bottom to top of dataset.
  - g.  $NumCTt$ : The number of CT wind grids.
6. ALLOCATE the following arrays:
- a.  $Tdata(NumCTt)$  (*CTWind*): The list of times for the CT-wind input files.
  - b.  $TimeStpCT(NumCTt)$  (*CTWind*): The list of time steps from the original LES simulation, associated with the CT-wind times.
7. Read the arrays from the CTS input file: DO\_LOOP IT=1,NumCTt
- a. READ (*UnWind*,\*,IOSTAT=*ErrStat*) *Tdata(IT)*, *TimeStpCT(IT)* and check no *ErrStat* errors as RETURN with  $NumCTt = IT - 1$
8. Close(*Unwind*), i.e. the *FileName* file

## **OUTPUT**

*CT\_SC\_ext*: C(3); extension of the turbulence file

It sets many other variables declared in parent module (*CTWind*): *InvMCTWS*, *CTYmax*, *CTZmax*, *CTDistSc* *CTLy*, *CTLz*, *NumCTt*, *Tdata(NumCTt)*, *TimeStpCT(NumCTt)*.

## L4. READCTSCALES(UNWIND, FILENAME, ERRSTAT)

Subroutine to read the input parameters for the coherent turbulence events, based on the large-eddy simulation (LES). Called by *CTInit* (*CTWind*). It reads scaling factors.

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Unit number for reading wind files
<i>FileName</i>	C(*)	IN	Name of the input CTTS wind file
<i>ErrStat</i>	I	OUT	If <>0, errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *NumCTy* (*CTWind*))

### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>I</i>	I			Loop counter

### STEPS

1. Open the *FileName* file with scales (LES or DNS) Call *OpenInpFile* (*UnWind*, TRIM( *FileName* ), *ErrStat*) (*NWTC\_IO* within *NWTC\_Library* package) and RETURN IF *ErrStat* <>0
2. Use *ReadCom* (*NWTC\_IO* within *NWTC\_Library* package) to read the first line header
3. Use *ReadVar* (*NWTC\_IO* within *NWTC\_Library* package) and READ the following variables all from (*CTWind*):
  - i. *CTVertShft*: Flag to indicate whether or not to shift the z values for the w component.
  - ii. DO\_LOOP *I*=1,3:
    1. *CTScale(I)*: Scaling factors to convert integer data to actual wind speeds.
    2. *CTOffset(I)*: Offsets to convert integer data to actual wind speeds.
  - iii. END\_DO
  - iv. *NumCTy*: The number of CT wind grid points in the y direction.
  - v. *NumCTz*: The number of CT wind grid points in the z direction.
4. Close(*Unwind*), i.e. the *FileName* file

### OUTPUT

*ErrStat*: *I*; the usual error flag

It sets many other variables declared in parent module (*CTWind*): *CTVertShft*, *CTScale(1:3)*, *CTOffset(1:3)*, *NumCTy*, *NumCTz*.

## L5. CT\_SETREFVAL(HEIGHT, HWIDTH, ERRSTAT)

Subroutine to that sets reference values for grid shift and reference height. Called by WindInf Init (*InflowWind*).

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type	INTENT	Description
<i>Height</i>	R(ReKi)	IN	Reference height (should be hub height)
<i>HWWidth</i>	R(ReKi)	IN, OPTIONAL	Reference offset (should be half grid width [~rotor radius])
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *CT\_Zref* (*CTWind*)).

#### LOCAL VARIABLES

N/A

#### STEPS

1. IF *TimeIndx* (*CTWind*) ==0 (not initialized) THEN:
  - a. warn user on the screen
  - b. Set *ErrStat*=1
  - c. RETURN
2. ELSE IF *CT\_Zref* (*CTWind*)  $\geq 0$  (The reference height for the CT file (the bottom of the billow); CTInit (*CTWind*) initializes it at -1.0) THEN :
  - a. warn user on the screen
  - b. Set *ErrStat*=1
  - c. RETURN
3. IF PRESENT(*HWWidth*) THEN: Set grid shift using the half-width:
  - a. *CTYHWid* (*CTWind*) = *HWWidth*
  - b. IF *CTYHWid* <0 THEN
    - i. warn user
    - ii. Set *CTYHWid* =0
    - iii. *ErrStat*=1
    - iv. RETURN
4. Set the reference height (bottom of the KH billow) using the input hub-height:
  - a.  $CT\_Zref = Height - CTZmax * CTLz$  (*CTWind*)
5. IF *CT\_Zref* (*CTWind*) <0 THEN :
  - a. warn user on the screen
  - b. Set *CT\_Zref*=0

c. Set *ErrStat* =1

## OUTPUT

*CT\_Zref (CTWind)*: R(ReKi): The reference height for the CT file (the bottom of the billow)

*CTYHWid (CTWind)*: R(ReKi): The half the width of the background dataset, used to compute the CTwind time offset

## L6. CT\_GETWINDSPEED (TIME, INPUTPOSITION, ERRSTAT)

This function returns the velocities at the specified time and space that are superimposed on a background wind flow. Called by WindInf\_getVelocity (InflowWind).

### THEORY

This function interpolates into the full-field CT wind arrays, performing a time shift based on the average windspeed. The modified time is used to decide which pair of time slices to interpolate within and between. After finding the two time slices, it decides which four grid points bound the (Y,Z) pair. It does a bilinear interpolation for (Y,Z) on each bounding time slice, then linearly interpolates between the 2 time slices. This routine assumes that X is downwind, Y is to the left when looking downwind and Z is up. In the time (X) and Z directions, steady winds are used when extrapolation is required. The dataset is assumed to be periodic in the Y direction.

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>Time</i>	R(ReKi)	IN	Time
<i>InputPosition(3)</i>	R(ReKi)	IN	Position (X, Y, Z)
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *NumCTt (CTWind)*)

#### LOCAL VARIABLES

Name	Type	Value Parameter/ Initialized	Description
<i>Iyz_th</i>	R(ReKi)		Temporary interpolated value. (time hi, all y, all z)
<i>Iyz_tl</i>	R(ReKi)		Temporary interpolated value. (time lo, all y, all z)
<i>Iyhz</i>	R(ReKi)		Temporary interpolated value. (y hi, all z)
<i>Iylz</i>	R(ReKi)		Temporary interpolated value. (y lo, all z)
<i>TimeShifted</i>	R(ReKi)		Shifted time (necessary because we're keeping x constant)
<i>Tgrid</i>	R(ReKi)		Fractional distance between time grids.
<i>Ygrid</i>	R(ReKi)		Fractional distance between grids in the y direction.
<i>Ynorm</i>	R(ReKi)		Nondimensional lateral distance of the analysis point from right side of dataset (looking downwind).

$Zgrid(3)$	R(ReKi)	Fractional distance between grids in the z direction.
$Znorm$	R(ReKi)	Nondimensional vertical distance of the analysis point from bottom of dataset.
$I$	I	Auxiliary index=1..3
$IYHi$	I	High index for interpolation in Y
$IYLo$	I	Low index for interpolation in Y
$IZHi(3)$	I	High indices for interpolation in Z
$IZLo(3)$	I	Low indices for interpolation in Z

### STEPS

1. Make sure that CTWind module is initialized and that the reference height is set:
  - a. IF  $TimeIndx (CTWind) == 0$  (not initialized) THEN:
    - i. warn user on the screen
    - ii. Set  $ErrStat=1$
    - iii. RETURN
  - b. ELSE IF  $CT\_Zref (CTWind) >= 0$  (The reference height for the CT file (the bottom of the billow); **CTInit** (**CTWind**) initializes it at -1.0) THEN :
    - i. warn user on the screen
    - ii. Set  $ErrStat=1$
    - iii. RETURN
2. Perform the time shift:  $TimeShifted = Time + (CTYHWid - InputPosition(1)) * InvMCTWS (CTWind)$ : here it is assumed that the coherent turbulence events are moving at MCTWS speed, and therefore the x-position is translated into a new time based on that translational velocity
3. Find the “time slices” that bound the shifted time so to linearly interpolate starting with a check that we are within bounds:
  - a. IF (  $TimeShifted <= Tdata(1) (CTWind)$  ) THEN Set  $TimeIndx=1$  and  $TGrid = 0.0$  ( $Tdata$  was read from input CTS file)
  - b. ELSEIF  $TimeShifted >= Tdata(NumCTt)$  THEN Set  $TimeIndx=NumCTt (CTWind)$  and  $TGrid = 1.0$
  - c. ELSE : prepare to interpolate
    - i. Set  $TimeIndx = \text{MAX}(\text{MIN}(TimeIndx, NumCTt-1), 1)$  (bound between 1 and  $NumCTt-1$ )
    - ii. DO\_LOOP: find the value and index of  $Tdata$  so that we are closest to  $TimeShifted$ 
      1. IF (  $TimeShifted < Tdata(TimeIndx) (CTWind)$  ) THEN  $TimeIndx = TimeIndx - 1$
      2. ELSEIF (  $TimeShifted >= Tdata(TimeIndx+1)$  ) THEN  $TimeIndx = TimeIndx + 1$

3. ELSE: here is the real interpolation to get the time fractional distance
  - a.  $Tgrid = \text{MIN}(\text{MAX}((TimeShifted - Tdata(TimeIndx)) / (Tdata(TimeIndx+1) - Tdata(TimeIndx)), 0.0), 1.0)$ ; TGrid becomes proportional to the distance of TimeShifted from Tdata(TimeIndx)
  - b. EXIT LOOP
  - iii. END\_DO
4. IF  $TimeStpCT(TimeIndx) (CTWind) == CTvel_files(2) (CTWind)$  THEN: Set  $IndCT\_lo = 2$  and  $IndCT\_hi = 1$  (notes: TimeStpCT was read from input CTS file; CTvel\_files was initialized in CTInit to 0)
5. ELSE:
  - a. Set  $IndCT\_lo = 1$  and  $IndCT\_hi = 2$
  - b. IF ( $TimeStpCT(TimeIndx) \neq CTvel\_files(IndCT\_lo)$ ) THEN:
    - i.  $CTvel\_files(IndCT\_lo) = TimeStpCT(TimeIndx)$
    - ii. Call ReadCTData ( CTWindUnit, CTvel\_files(IndCT\_lo), IndCT\_lo, ErrStat ) (CTWind)
6. Calculate the y values now, in a similar way to time slices, noting that there is periodicity of KH data in this direction. Also the lower-right corner is (1,1) when looking downwind, so find normalized distance along y of the current point (adjust for shift in reference frame associated with tower position): (extra parameters initialized in CTInit (CTWind)) :
  - a.  $Ynorm = (CTYt + InputPosition(2)) / CTYmax (CTWind)$ ; normalized y-distance
  - b. IF  $Ynorm < 0$  THEN  $Ynorm = 1.0 + \text{MOD}(Ynorm, 1.0)$  (periodicity)
  - c.  $Ygrid = \text{MIN}(\text{MAX}(\text{MOD}(Ynorm, DelYCTgrid), 0.0), 1.0)$  (CTWind)
  - d.  $IYLo = \text{MAX}(\text{MOD}(\text{INT}(Ynorm * NumCTyD1) + 1, NumCTyD1), 1)$ ; low index
  - e.  $IYHi = \text{MOD}(IYLo, NumCTyD) + 1$ ; high index
7. Calculate the z values now, in a similar way to y slices; however note the equivalent Znorm for the w-component may be shifted vertically by half the original grid spacing. (the K-H data staggers w differently than u & v). We store IZLo, IZHi, and Zgrid in an array to account for this difference:
  - a.  $Znorm = \text{MIN}(\text{MAX}((InputPosition(3) - CT_Zref) / CTZmax, 0.0), 1.0)$  (CTWind); non-dimensional height (CT\_Zref is the bottom of the billow)
  - b.  $Zgrid(1:2) = \text{MIN}(\text{MAX}(\text{MOD}(Znorm, DelZCTgrid), 0.0), 1.0)$  (CTWind) assign same value to first two array elements, this is so that we can use the same interpolation scheme and equations at the end
  - c.  $IZLo(1:2) = \text{MAX}(\text{INT}(Znorm * NumCTzD1 (CTWind)) + 1, 1)$
  - d. IF ( $IZLo(1) == NumCTzD$ ) THEN (at upper end of z dimension):
    - i.  $IZLo(1:2) = NumCTzD1$
    - ii.  $Zgrid(1:2) = 1.0$
8. Find the equivalent Znorm for the w-component, which may be shifted vertically by half the original grid spacing. LES and DNS scale differently:
  - a. IF ( $CTVertShft$ ) (CTWind; read from the CTS scale file see ReadCTScales) THEN:

- i.  $Znorm = \text{MAX}( Znorm - 0.5 * \text{DelZCTgrid} / CT\_DF\_Z(CTWind), 0.0 )$
  - ii.  $Zgrid(3) = \text{MIN}( \text{MAX}( \text{MOD}( Znorm, \text{DelZCTgrid} ), 0.0 ), 1.0 )$
  - iii.  $IZLo(3) = \text{MAX}( \text{INT}( Znorm * \text{NumCTzD1} ) + 1, 1 )$ ; Make sure the lowest possible value is 1.
  - iv. IF ( $IZLo(3) == \text{NumCTzD}$ ) THEN (at upper end of z dimension):
    - 1.  $IZLo(3) = \text{NumCTzD1}$
    - 2.  $Zgrid(3) = 1.0$
  - b. ELSE ( $CTVertShft = \text{.FALSE.}$ )
    - i.  $IZLo(3) = IZLo(1)$
    - ii.  $Zgrid(3) = Zgrid(1)$
9. Set  $IZHi(:) = IZLo(:) + 1$
10. Interpolate for [U/V/W] component of wind velocity within the grid, by first interpolating in the lower time slice along z and then along y; then the upper time slice along z and then along y; then interpolate in time
- a.  $I=[1/2/3]$
  - b.  $Iylz = ( CTvel[U/V/W](IYLo,IZHi(I),IndCT_lo) - CTvel[U/V/W](IYLo,IZLo(I),IndCT_lo) ) * Zgrid(I) + CTvelU(IYLo,IZLo(I),IndCT_lo)$  (along z, lower time, lower y)
  - c.  $Iyhz = (CTvel[U/V/W](IYHi,IZHi(I),IndCT_lo) - CTvel[U/V/W](IYHi,IZLo(I),IndCT_lo)) * Zgrid(I) + CTvel[U/V/W](IYHi,IZLo(I),IndCT_lo)$  (along z, lower time, higher y)
  - d.  $Iyz_tl = (Iyhz - Iylz) * Ygrid + Iylz$  (along y, lower time)
  - e.  $Iylz = (CTvel[U/V/W](IYLo,IZHi(I),IndCT_hi) - CTvel[U/V/W](IYLo,IZLo(I),IndCT_hi)) * Zgrid(I) + CTvelU(IYLo,IZLo(I),IndCT_hi)$  (along z, higher time, lower y)
  - f.  $Iyhz = (CTvel[U/V/W](IYHi,IZHi(I),IndCT_hi) - CTvel[U/V/W](IYHi,IZLo(I),IndCT_hi)) * Zgrid(I) + CTvel[U/V/W](IYHi,IZLo(I),IndCT_hi)$  (along z, higher time, higher y)
  - g.  $Iyz_th = (Iyhz - Iylz) * Ygrid + Iylz$  (along y, higher time)
  - h.  $CT\_GetWindSpeed\%Velocity(I) = (Iyz_th - Iyz_tl) * Tgrid + Iyz_tl$  (along time)

## OUTPUT

*ErrStat: I;* the usual error flag.

*CT\_GetWindSpeed:* TYPE(**InflIntrpOut**) (*SharedInflowDefns*); the wind velocity (no tower effect) at the point of interest following the CT file type, which is a combination of Kelvin-Helmotz billows superimposed on a background wind file.

## L7. READCTDATA (UNWIND, CTFILENO, ITIME, ERRSTAT)

This subroutine is used to read one time-step's worth of large-eddy (LES) zero-mean wind data for each wind component from a file. Called by *CT\_GetWindSpeed(CTWind)*.

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Input File unit number
<i>CTFileNo</i>	I	IN	Number of file to read
<i>Itme</i>	I	IN	Index of time slice
<i>ErrStat</i>	I	OUT	If <>0, errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *CTSpah* (*CTWind*)).

### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>CTNum</i>	C(5)			String equivalent of input variable CTFFileNo
<i>FileName</i>	C(1024)			The name of the input data file

### STEPS

1. IF *CTFileNo* == 0 THEN set *CTvel[U/V/W](::,Itme)* =0.0 (*CTWind*)
2. ELSE:
  - a. Put *CTFileNo* into a string into *CTNum*
  - b. Set *filename* = TRIM( *CTSpah* (*CTWind*) )//'\u\u\_16i\_//*CTnum*//'.'//TRIM( *CText* (*CTWind*) )
  - c. Get the velocity components from the file: Call *LoadCTData(UnWind, TRIM(FileName), Itme, [1/2/3], CTvel[U/V/W], ErrStat)* (*CTWind*)

### OUTPUT

*ErrStat*: I; the usual error flag.

It also sets the *CTvel[U/V/W]* (*CTWind*) velocity components from the full field CT file.

## L8. LOADCTDATA (UNWIND,FILENAME,ITIME,ICompm,VEL,ERRSTAT)

This subroutine is used to read velocity from the turbulent event file at the y and z grid points (decimated) and for the time under consideration. Called by [ReadCTData](#) (*CTWind*). Could this one be included in [ReadCTData](#)?

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>UnWind</i>	I	IN	Input File unit number
<i>FileName</i>	C(*)	IN	The name of the file to open
<i>Itme</i>	I	IN	Index of time slice
<i>IComp</i>	I	IN	Index of velocity component
<i>Vel(NumCTyD,NumCTzD,2)</i>	R(ReKi)	INOUT	It returns the velocity array
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *NumCTy* (*CTWind*)).

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>Com(NumCTy)</i>	I(B2Ki)			Temporary array to hold component's integer values for a given Z
<i>IY</i>	I			A loop index for indexing the arrays in the y direction
<i>IYK</i>	I			An index for the decimated arrays in the y direction
<i>IZ</i>	I			A loop index for indexing the arrays in the z direction
<i>IZK</i>	I			An index for the decimated arrays in the z direction

#### STEPS

1. Open the input file: CALL [OpenUIInBEFile](#)( *UnWind*, TRIM(*FileName*), 2\**NumCTy*, *ErrStat* ) (*NWTC\_IO* within *NWTC\_Library*) (big endian unformatted file\_

2. Initialize  $IZK=0$  (the Z index into the array (necessary b/c of decimation factor))
3. DO\_LOOP  $IZ=1, NumCTz, CT\_DF\_Z (CTWind)$  : read the file
  - a. READ ( $UnWind, REC=IZ, IOSTAT=ErrStat$ ) Com
  - b.  $IZK = IZK + 1$
  - c. Initialize  $IYK = 0$  (the Y index into the array (necessary b/c of decimation factor))
  - d. DO\_LOOP  $IY=1, NumCTy, CT\_DF\_Y (CTWind)$ 
    - i.  $IYK = IYK + 1$
    - e.  $Vel(IYK, IZK, ITime) = CTScale(IComp) * Com(IY) + CTOffset(IComp) (CTWind)$
    - f. END\_LOOP
4. END\_LOOP
5. Close ( $UnWind$ ) File

## **OUTPUT**

*ErrStat: I; the usual error flag.*

*Vel(NumCTyD, NumCTzD, 2): R(ReKi) ; velocity components at the time requested along the decimated y and z grid points from the turbulent event file.*

## L9. CT\_TERMINATE (ERRSTAT)

Subroutine to close files, deallocate memory, and un-set the initialization flag. Called by *CTInit (CTWind)* and *WindInf Init (InflowWind)*.

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *CTWindUnit (CTWind)*)

### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>I</i>	I			Loop counter

### STEPS

1. Close file unit *CTWindUnit*: unit number used to read the wind files at each call to *CT\_GetWindSpeed()*
2. DEALLOCATE if allocated :
  - a. *CTvelU, CTvelV, CTvelW, Tdata, TimeStpCT (CTWind)*
3. Set *TimeIndx=0*; initialization flag: if 1=initialized

### OUTPUT

*ErrStat*: *I*; the usual error flag

It closes files and deallocates memory.

## APPENDIX M. USERWIND MODULE

### GENERAL ORGANIZATION

The purpose of this module is to allow user-defined wind. For this reason, it is a placeholder for user written code. As it is, i.e. coming from NREL, it is mostly blank, and it uses a simple steady state, uniform wind.

### EXTERNAL MODULES

- *NWTC\_Library* [see NWTC\_Library.f90]
- *SharedInflowDefns* [see SharedInflowDefs.f90]

### PRIVATE (LOCALLY DECLARED) DATA

Parameter				
Name	Type	Value	/	Description
<b>Initialized</b>				
<i>Initialized</i>	L	.FALSE.	Initialized, SAVE	Whether or not the initialization routine has been run
<i>UWmeanU</i>	R(ReKi)			Possibly instantaneous, disk-averaged wind speeds
<i>UWmeanV</i>	R(ReKi)			Possibly instantaneous, disk-averaged wind speeds
<i>UWmeanW</i>	R(ReKi)			Possibly instantaneous, disk-averaged wind speeds

### PUBLIC (LOCALLY DECLARED) DATA

- New PUBLIC Routines and Functions

Name	Description
<u><i>UsrWnd_Init</i></u>	Initialization of the module and read input
<u><i>UsrWnd_GetValue</i></u>	To get values associated with variable names
<u><i>UsrWnd_GetWindSpeed</i></u>	Wind Velocity at time and position of interest
<u><i>UsrWnd_Terminate</i></u>	Terminate (Deallocate) module

## M1. USRWND\_INIT (ERRSTAT)

Subroutine called at the beginning of a simulation. The user is supposed to edit it so to make use of the desired wind input. Called by *WindInf Init* (*InflowWind*).

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *TimeIndx* (*SharedInflowDefns*)).

#### LOCAL VARIABLES

N/A

#### STEPS

1. IF *Initialized* (parent module *UserWind*): module already initialized: THEN:
  - a. warn user and RETURN with *ErrStat*=1
2. ELSE:
  - a. *ErrStat*=0 and Call *NWTC Init* (*NWTC\_Library*)
3. Write on screen that User-Defined Wind is used
4. Set *UWmean[U/V/W]* to a hardwired user data (e.g. *UWmean[U/V/W]*=[10.0 , 0.0, 0.0]
5. Set *Initialized* = .TRUE.

#### OUTPUT

*ErrStat* : I; usual error status flag.

*UWmean[U/V/W]*: R(ReKi) (*UserWind*); velocity set by user.

It also sets *Initialized*: L; (*UserWind*); initialization flag.

## M2. USRWND\_GETVALUE(VARNAME, ERRSTAT)

Function that returns a real scalar value whose name is listed in the VarName input argument. If the name is not recognized, an error is returned in ErrStat. Called by WindInf\_Adhack\_diskVel (*InflowWind*).

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90].

#### INPUT

Name	Type/Module	INTENT	Description
<i>VarName</i>	C(*)	IN	Variable name string
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Initialized* (*UsrWind*)).

#### LOCAL VARIABLES

Name	Type/Module	Value	Parameter/ Initialized	Description
<i>VarNameUC</i>	C(20)			Upper case <i>VarName</i>

#### STEPS

1. IF .NOT. *Initialized* (parent module *UserWind*): module NOT initialized: THEN:
  - a. warn user and RETURN with *ErrStat*=1
2. ELSE initialize *ErrStat*=0
3. Transform to upper case *VarNameUC* = *VarName*
4. Based on *VarNameUC*:
  - a. 'MEAN[U/V/W]'  $\rightarrow$  *UsrWnd\_GetValue* = *UWmean*[U/V/W]
  - b. All other cases: warn user of invalid variable name and set *ErrStat*=1

#### OUTPUT

*UsrWnd\_GetValue*: R(ReKi); the value of the variable whose name string was input.

*ErrStat*: I; the usual error flag.

### M3. USRWND\_GETWINDSPEED (TIME, INPUTPOSITION, ERRSTAT)

This function receives time and position (in *InputInfo*) where (undisturbed) velocities are requested. It returns the velocities at the specified time and space. The user is supposed to edit it so to make use of the desired wind input. Called by WindInf\_getVelocity (*InflowWind*).

#### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

#### INPUT

Name	Type/Module	INTENT	Description
<i>Time</i>	R(ReKi)	IN	Time from the start of the simulation
<i>InputPosition</i> (3)	R(ReKi)	IN	Position (X, Y, Z)
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Initialized* (*UserWind*)).

#### LOCAL VARIABLES

N/A

#### STEPS

1. IF .NOT. *Initialized* (parent module *UserWind*): module NOT initialized: THEN:
  - a. warn user and RETURN with *ErrStat*=1
2. ELSE initialize *ErrStat*=0
3. The function returns a steady and uniform wind for the time being:
  - a. *UsrWnd\_GetWindSpeed%Velocity*(1) = 10.0 ; U velocity (along positive X)
  - b. *UsrWnd\_GetWindSpeed%Velocity*(2) = 0.0 ; V velocity (along positive Y)
  - c. *UsrWnd\_GetWindSpeed%Velocity*(3) = 0.0 ; W velocity (along positive Z)

#### OUTPUT

*ErrStat*: I; the usual error flag.

*UsrWnd\_GetWindSpeed*: TYPE(**InflIntrpOut**) (*SharedInflowDefns*); the wind velocity at the point of interest following the user type wind data file.

## M4. USRWND\_TERMINATE (ERRSTAT)

Subroutine to close files, deallocate memory, and un-set the initialization flag. The user is supposed to edit it so to make use of the desired wind input. Called by WindInf\_Terminate (InflowWind).

### EXTERNAL (INVOKED) MODULES

From Parent Module *InflowWind*[Mod.f90]

### INPUT

Name	Type/Module	INTENT	Description
<i>ErrStat</i>	I	OUT	If $\neq 0$ , errors encountered

Additional input is given through parent module variables and invoked modules in the parent module (e.g. *Initialized (UserWind)*).

### LOCAL VARIABLES

N/A

### STEPS

1. Set *ErrStat*=0
2. Set *Initialize*=.FALSE.; initialization flag: if T=initialized

### OUTPUT

*ErrStat*: I; the usual error flag

It should also deallocate memory, close any file that the user may have dealt with.

## APPENDIX N. DIRECTION COSINES

It is worth spending a few words illustrating the role of the rotational matrices used in AeroDyn to go from one reference frame to another.

The (direction-cosine) matrix  $[D_c]$  allows to go from global to the generic local reference system:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = [D_c] \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

$[D_c]$  can be seen as follows: rows 1 through 3 represent the components along the global frame of reference of the local  $\hat{i}$ ,  $\hat{j}$ ,  $\hat{k}$  respectively. In effect, row 1 through 3 contain the direction cosines of the vectors  $\hat{i}$ ,  $\hat{j}$ ,  $\hat{k}$ , respectively.

$[D_c]^{-1} = [D_c]^T$  and allows to go from the local to the global reference system, and it is equivalent to considering a rotation with angles of opposite signs w.r.t. those relative to a rotation from local to global frames.

Furthermore: consider wanting to rotate a vector  $\underline{U} = \begin{bmatrix} U_x \\ U_y \\ U_z \end{bmatrix}$  by a certain angle about a certain axis. This is

equivalent to rotating the reference frame (O;X,Y,Z) by the same angle but with opposite sign. So

$$\underline{u} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = [D_c]^T \underline{U} = [D_c]^T \begin{bmatrix} U_x \\ U_y \\ U_z \end{bmatrix} \text{ is the rotated vector.}$$

Now consider the following problem: calculate the component on the rotation plane (~rotor) of the wind velocity  $\underline{V}$ , while the local blade element coordinates (x,y) are known, as are the orientation ( $\hat{i}$ ,  $\hat{j}$ ) and the pitch  $\theta$ . ( $\hat{i}$ ,  $\hat{j}$ ) are assumed to be oriented with x along chord pointing towards LE, and y upward towards the camber apex. (This is not the Aerodyn standard, see below).

One could first take the components of  $\underline{V}$  along the local frame ( $\hat{i}$ ,  $\hat{j}$ ) and then decompose those in the rotation plane:

$$\underline{V} = \begin{bmatrix} V_x \hat{i} \\ V_y \hat{j} \\ V_z \hat{k} \end{bmatrix} = V_x \hat{i} + V_y \hat{j} + V_z \hat{k} = [V_x \quad V_y \quad V_z] \begin{bmatrix} \hat{i} \\ \hat{j} \\ \hat{k} \end{bmatrix} = \begin{bmatrix} (\underline{V} \cdot \hat{i}) \hat{i} \\ (\underline{V} \cdot \hat{j}) \hat{j} \\ (\underline{V} \cdot \hat{k}) \hat{k} \end{bmatrix}$$

Consider the rotation around the local z-axis of the local x-axis into the plane of rotation, by way of  $\theta$  angle, then the component of  $\underline{V}$  along the new x,  $V_T$ , can be found as:

$$V_T = V_x \cos(\theta) + V_y \sin(\theta) = \underline{V} \cdot \hat{i} \cos(\theta) + \underline{V} \cdot \hat{j} \sin(\theta) = \underline{V} \cdot (\hat{i} \cos(\theta) + \hat{j} \sin(\theta))$$

Note that:  $\hat{i} = [D_c]_{1,:}$ ,  $\hat{j} = [D_c]_{2,:}$ , and  $\hat{k} = [D_c]_{3,:}$

So one could think of constructing a temporary vector:

$$(\hat{i} \cos(\theta) + \hat{j} \sin(\theta)) = (\cos(\theta) [D_c]_{1,:} + \sin(\theta) [D_c]_{2,:})$$

and then doing a DOT\_PRODUCT of that vector and  $\underline{V}$ . This geometrical trick is used in AeroDyn at times, as in [AD\\_CalculateLoads](#), with the only difference that the (x,y) local reference frame is oriented differently from what is assumed here, with y along the chord pointing towards the TE, and x towards the camber apex. However, the local frame  $(\hat{i}, \hat{j})$  is not oriented normal to the plane of rotation if there is a coning angle. In fact  $\hat{k}$  is oriented along the blade z axis not necessarily belonging to the plane of rotation.