```
!module that defines the input variables without specifying the size, for using them all the code along
module geometric_inputs

    integer,save::n_point, n_blades
    real, allocatable, save::geom(:,:)
    real :: hub_to_br, hub_height, tilt_angle
!validated
end module geometric_inputs
!-------------------------------------------------------
!module with the characteristics positions of the horseshoes and the control points
module horseshoe_locations
    integer,save::n_horseshoe
    real (kind=8),allocatable,save::horseshoe_loc(:,:),horseshoe_te_loc(:,:),
    control_points_loc(:,:),area_distr(:),chord_distr(:),twist(:),thickness(:)
    real (kind=8),allocatable,save::hs_loc_pol(:,:),cp_loc_pol(:,:)
    real (kind=8),allocatable,save::temp_hs_loc(:,:,:),temp_cp_loc(:,:,:),temp_hs_te_loc(:,
    :,:), cp_saved_pos(:,:,:)
    real (kind=8),allocatable,save::el_vect1(:,:,:),el_vect2(:,:,:),el_vect3(:,:,:)

end module horseshoe_locations
!-------------------------------------------------------
!module with the variables related to the time simulation
module clock
    integer,save::n_time_steps, time_step
    real,save::inc_time,time_sample
    real, allocatable, save:: time(:)

end module clock
!-------------------------------------------------------
!module containing some operating parameters of the turbine and external conditions
module operating_parameters
    real,save::omega, v0(3), density, upflow_angle, yaw_misalignment, wind_speed_module,
    exp_a

end module operating_parameters
!-------------------------------------------------------
!module for the aerodynamic variables and coefficients
module aerodynamic_performance
    real,save::azimut,pitch
    real, save:: preazimut
    real (kind=8),allocatable,save::alpha(:,:),Cl(:,:),Cd(:,:),Cm(:,:)
    real, allocatable, save:: L(:,:),D(:,:),T(:),Q(:),P(:),Mz(:,:),Cp(:),Ct(:),MB_ip(:,:),
    MB_op(:,:)

end module aerodynamic_performance
!-------------------------------------------------------
!Module dedicated to the aerodynamic coefficients of the introduced airfoils
module airfoil_polars
    integer,save:: n_data_points, n_airfoils
    real (kind=8),allocatable,save:: Cl_pol(:,:),Cd_pol(:,:),Cm_pol(:,:)
    real(kind=8), allocatable,save:: AOA(:,:),Cl_pol_aux(:,:),Cd_pol_aux(:,:),Cm_pol_aux(:,
    :)
    real,allocatable,save:: airfoils_thickness(:)

end module airfoil_polars
!-------------------------------------------------------
!
module incident_speeds
    real (kind=8),allocatable,save:: wake_ind(:,:,:),wind(:,:,:),turning_ind(:,:),blade_ind
    (:,:,:),total_wind(:,:,:),blade_ind2(:,:)
    !test
    real (kind=8),allocatable,save:: old_wake_ind(:,:),test_blade_ind(:,:),wake_ind2(:,:,:)
    ,wake_ind3(:,:)

end module incident_speeds
!-------------------------------------------------------
module vortex_intensities
    real (kind=8),allocatable,save:: pr_attached_hs_int(:,:), pr_wake_int(:,:)

end module vortex_intensities
!-------------------------------------------------------
module wake_sheet
    integer,save:: n_wake_lines    !the total number of nodes in the wake sheet will be
```

```
    n_wake_lines*n_horseshoe
    real (kind=8), save:: wake_x_spacing
    real (kind=8),allocatable,save:: pr_wake_nodes_loc(:,:,:)
    integer, allocatable, save:: wake_nodes_connec(:,:)
    real (kind=8),allocatable,save:: wake_segments_int_s(:,:),wake_segments_int_c(:,:)

end module wake_sheet
!-----------------------------------------------------
module tf_matrix
    real (kind=8),allocatable,save:: attached_s_matrix_u(:,:),attached_s_matrix_v(:,:),
    attached_s_matrix_w(:,:),attached_c_matrix_u(:,:),attached_c_matrix_v(:,:),
    attached_c_matrix_w(:,:)
    real (kind=8),allocatable,save:: wake_s_matrix_u(:,:,:),wake_s_matrix_v(:,:,:),
    wake_s_matrix_w(:,:,:),wake_c_matrix_u(:,:,:),wake_c_matrix_v(:,:,:),wake_c_matrix_w(:
    ,:,:)

end module tf_matrix
!-----------------------------------------------------
module iteration_data
    real  (kind=8),allocatable,save:: aux_attached_hs_int(:),pr_cycle_att_hs_int(:,:)
end module iteration_data
!-----------------------------------------------------
module simulation_data
    integer,save:: wake_int_init,wake_loc_init, b_wake_roll_up, b_allow_wake_evol,
    b_stationary_sim
    integer, allocatable,save:: blade_phase(:,:)
end module simulation_data
!-----------------------------------------------------
module lambda_variables
    real, allocatable:: lambda(:),cp_lambda(:),ct_lambda(:)
end module lambda_variables
!-----------------------------------------------------
module power_curve_variables
    real,allocatable:: pc_P(:),pc_T(:),pc_omega(:),pc_wspeedm(:)
    integer n_pc_steps
end module power_curve_variables
!-----------------------------------------------------
module file_names
    character(100) bladeplanform, airfoilpolars, inputfile
end module file_names
!-----------------------------------------------------
program hub

    use file_names
    implicit none
    real t1
    INTEGER :: count1, count_rate, count_max, count2
    integer n,m
    count_rate=1
    CALL SYSTEM_CLOCK(count1, count_rate, count_max)

    print*,"Please, write the name of the inputs file"
    read(*,*) inputfile
    print*, inputfile,"OK, simulation will begin"
    open(UNIT=7,FILE=inputfile,ACCESS='SEQUENTIAL')
    read(7,*) n
    read(7,*) bladeplanform
    read(7,*) airfoilpolars
    close(7)

    select case(n)
    case(1)
        call single_point_simulation
    case(2)
        call full_cycle_simulation
    case(3)
        call find_optimal_lambda
    case(4)
        call power_curve
    end select

    CALL SYSTEM_CLOCK(count2, count_rate, count_max)
    t1=(count2-count1)/count_rate
    open(UNIT=7,FILE='Results_Simulation_time.txt',ACCESS='SEQUENTIAL')
```

```
        write(7,*) t1
    close(7)
    print*,"Simulation took ",t1, " seconds. Results are ready."

    pause

end program
!-------------------------------------------------------
!third version of the subroutine that calculates the induced speed at a certain point of
    the 3D space by a vortex line segment defined by two points
subroutine VLtfnc(x,y,z,x1,y1,z1,x2,y2,z2,gam,u,v,w)
    implicit none
    real (kind=8),intent(in):: x,y,z,x1,y1,z1,x2,y2,z2,gam
    real (kind=8),intent(out):: u,v,w
    real (kind=8):: r1(3),r2(3),r0(3),magr1,magr2,magr0
    real (kind=8):: aux_cross(3),aux_dot,aux_sp(3)
    real,parameter:: dist=0.025, pi=4*atan(1.)

    r1(1)=x-x1
    r1(2)=y-y1
    r1(3)=z-z1
    r2(1)=x-x2
    r2(2)=y-y2
    r2(3)=z-z2
    r0(1)=x2-x1
    r0(2)=y2-y1
    r0(3)=z2-z1

    magr1=sqrt(dot_product(r1,r1))
    magr2=sqrt(dot_product(r2,r2))
    magr0=sqrt(dot_product(r0,r0))
    aux_dot=dot_product(r1,r2)
    call crossprod(r1,r2,aux_cross)

    aux_sp=(gam/(4*pi))*((magr1+magr2)*aux_cross)/(magr1*magr2*(magr1*magr2+aux_dot)+(dist
    *magr0)**2)

    u=aux_sp(1)
    v=aux_sp(2)
    w=aux_sp(3)

!created 12/01/2013
!this transfer function has been taken from the ECN report
!validated
end subroutine VLtfnc
!-------------------------------------------------------------------------------------------
    --
!Alternative version, based on the theory presented in Low Speed Aerodynamics
subroutine VLtfnc_LSA(x,y,z,x1,y1,z1,x2,y2,z2,gam,u,v,w)
    !Low Speed Aerodynamics--> LSA
    implicit none
    real (kind=8),intent(in):: x,y,z,x1,y1,z1,x2,y2,z2,gam
    real (kind=8),intent(out):: u,v,w
    real (kind=8):: r1(3),r2(3),r0(3),magr1,magr2,magr0,magrx
    real (kind=8):: aux_cross(3),aux_dot,aux_sp(3),aux_dot1,aux_dot2
    real,parameter:: tol=0.0000003, pi=4*atan(1.)

    r1(1)=x-x1
    r1(2)=y-y1
    r1(3)=z-z1
    r2(1)=x-x2
    r2(2)=y-y2
    r2(3)=z-z2
    r0(1)=x2-x1
    r0(2)=y2-y1
    r0(3)=z2-z1

    magr1=sqrt(dot_product(r1,r1))
    magr2=sqrt(dot_product(r2,r2))
    magr0=sqrt(dot_product(r0,r0))
    call crossprod(r1,r2,aux_cross)
    aux_dot=dot_product(aux_cross,r0)
    aux_dot1=dot_product(r0,r1)
    aux_dot2=dot_product(r0,r2)
```

```
    magrx=dot_product(aux_cross,aux_cross)

    if ((magr1.LT.tol).OR.(magr2.LT.tol).OR.(magrx.LT.tol)) then
        u=0.d0
        v=0.d0
        w=0.d0
    else
        aux_sp=(gam/(4*pi))*aux_cross*((aux_dot1/magr1)-(aux_dot2/magr2))/magrx
        u=aux_sp(1)
        v=aux_sp(2)
        w=aux_sp(3)
    end if


!created 20/04/2013
!this transfer function has been taken from Low Speed Aerodynamics book
!validated, not used
!Possible alternative
end subroutine VLtfnc_LSA
!-------------------------------------------------------------------------------
!subroutine that calculates the cross-product of two given vectors
subroutine crossprod(a,b,c)

    implicit none
    real (kind=8), intent(in)::a(3),b(3)
    real (kind=8),intent(out)::c(3)

    c(1)=a(2)*b(3)-a(3)*b(2)
    c(2)=a(3)*b(1)-a(1)*b(3)
    c(3)=a(1)*b(2)-a(2)*b(1)
!validated 3/11/2012
end subroutine crossprod
!-----------------------------------------------------
!subroutine that reads the input data (wing planform)
subroutine read_input()
    use geometric_inputs
    use airfoil_polars
    use horseshoe_locations
    use file_names
    implicit none
    integer i,j,max_n_data
    integer,allocatable ::n_data_input(:)

    real, parameter:: pi=4*atan(1.)
    real,allocatable::aux(:,:)
    real, allocatable:: aux_alpha(:,:),aux_cl(:,:),aux_cd(:,:),aux_cm(:,:)
    open(UNIT=5,FILE=bladeplanform,ACCESS='SEQUENTIAL')
    read(5,*) n_point
    allocate(geom(5,n_point))
    do i=1,n_point
        read(5,*) geom(:,i)
    end do
    close(5)
    !if twist has been introduced in degrees uncomment next line
    geom(4,:)=geom(4,:)*pi/180.
    open(UNIT=5,FILE=airfoilpolars,ACCESS='SEQUENTIAL')

    read(5,*) n_airfoils
    allocate(n_data_input(n_airfoils))
    max_n_data=1
    do i=1,n_airfoils
        read(5,*) n_data_input(i)
        if (n_data_input(i).GT.n_data_input(max_n_data)) then
            max_n_data=i
        end if
    end do

    allocate(airfoils_thickness(n_airfoils))
    allocate(aux_alpha(n_airfoils,n_data_input(max_n_data)))
    allocate(aux_cl(n_airfoils,n_data_input(max_n_data)))
    allocate(aux_cd(n_airfoils,n_data_input(max_n_data)))
    allocate(aux_cm(n_airfoils,n_data_input(max_n_data)))

    do i=1,n_airfoils
```

```
        read(5,*) airfoils_thickness(i)
        do j=1,n_data_input(i)
            read(5,*) aux_alpha(i,j),aux_cl(i,j),aux_cd(i,j),aux_cm(i,j)
            !activate the next line if the angle of attack has been introduced in degrees
            aux_alpha(i,j)=pi*aux_alpha(i,j)/180
        end do
    end do
    close(5)


    n_data_points=n_data_input(max_n_data)
    allocate(AOA(n_airfoils,n_data_points))
    allocate(Cl_pol_aux(n_airfoils,n_data_points))
    allocate(Cd_pol_aux(n_airfoils,n_data_points))
    allocate(Cm_pol_aux(n_airfoils,n_data_points))
    allocate(Cl_pol(n_horseshoe,n_data_points))
    allocate(Cd_pol(n_horseshoe,n_data_points))
    allocate(Cm_pol(n_horseshoe,n_data_points))
    allocate(aux(4,n_data_points))

    do i=1,n_airfoils
        AOA(i,:)=aux_alpha(max_n_data,:)
        do j=1,n_data_points
            call spline(aux_alpha(i,1:n_data_input(i)),aux_cl(i,1:n_data_input(i)),AOA(i,
    j),Cl_pol_aux(i,j),n_data_input(i),1)
            call spline(aux_alpha(i,1:n_data_input(i)),aux_cd(i,1:n_data_input(i)),AOA(i,
    j),Cd_pol_aux(i,j),n_data_input(i),1)
            call spline(aux_alpha(i,1:n_data_input(i)),aux_cm(i,1:n_data_input(i)),AOA(i,
    j),Cm_pol_aux(i,j),n_data_input(i),1)
        end do

    end do

!validated 3/11/2012
!modified 13/04/2013
end subroutine read_input
!-----------------------------------------------------------
!subroutine for calculating the components of the Hermite polynomial
subroutine hermite_poly(chi,poly)
    implicit none
    real, intent(in)::chi
    real, intent(out)::poly(4)
    poly(1)=(1+chi+chi)*(chi-1)*(chi-1)
    poly(2)=chi*(chi-1)*(chi-1)
    poly(3)=chi*chi*(3-2*chi)
    poly(4)=(chi-1)*chi*chi
!validated
end subroutine
!-----------------------------------------------------------
!subroutine for calculating the first derivate values of a discrete defined function at
    selected points
subroutine derivate(x,f,der,n)

    implicit none
    integer,intent(in)::n
    real,intent(in)::x(n),f(n)
    real (kind=8),intent(out)::der(n)
    real der_aux(n-1),aux1,aux2
    integer i

    do i=1,n-1
        der_aux(i)=(f(i+1)-f(i))/(x(i+1)-x(i))
    end do
    der(1)=der_aux(1)
    der(n)=der_aux(n-1)

    do i=2,n-1
        aux1=(x(i)+x(i-1))/2
        aux2=(x(i+1)+x(i))/2
        der(i)=der_aux(i-1)+(der_aux(i)-der_aux(i-1))*(x(i)-aux1)/(aux2-aux1)
    end do
!validated 3/11/2012
end subroutine derivate
!-----------------------------------------------------
```

```
!subroutine that interpolates the value of a discretely defined function, 2 points given
subroutine interpolate(x1,f1,deriv,x2,f2)

    implicit none
    real,intent(in)::x1(2),f1(2)
    real (kind=8),intent(in)::x2,deriv(2)
    real (kind=8),intent(out)::f2
    real chi,poly(4)
    chi=(x2-x1(1))/(x1(2)-x1(1))
    call hermite_poly(chi,poly)
    f2=poly(1)*f1(1)+poly(3)*f1(2)+(poly(2)*deriv(1)+poly(4)*deriv(2))*(x1(2)-x1(1))
!validated 3/11/2012
end subroutine interpolate
!-----------------------------------------------------
!subroutine for obtaining the values of a discretely defined function in all the points ↙
    required
subroutine spline(x1,f1,x2,f2,n1,n2)

    implicit none
    integer,intent(in)::n1,n2
    real,intent(in)::x1(n1),f1(n1)
    real (kind=8),intent(in)::x2(n2)
    real (kind=8),intent(out)::f2(n2)
    integer i,j,point
    real(kind=8) der(n1)

    !first it is necessary to check if the inputs are correct
    do i=1,n1-1
        if(x1(i).GT.x1(i+1)) then
            print*,'The points where the function is defined are not consecutive'
            print*, x1(i),x1(i+1)
            pause
            stop
        end if
    end do
    do i=1,n2-1
        if(x2(i).GT.x2(i+1)) then
            print*, 'The points where the function is wanted to be defined are not ↙
    consecutive'
            pause
            stop
        end if
    end do

    if((x2(1).LT.x1(1)).OR.(x2(n2).GT.x1(n1))) then
        print*,'The new points are out of the defined range of points'
        print*,x2(1),x1(1),x1(n1),x2(n2)
        pause
        stop
    end if
    call derivate(x1,f1,der,n1)

    point=1
    do i=1,n2
        do j=point,n1-1
            if(x1(j+1).GT.x2(i)) exit
        end do
        point=j
        call interpolate(x1(j:j+1),f1(j:j+1),der(j:j+1),x2(i),f2(i))
    end do
!validated 3/11/2012
end subroutine spline
!-----------------------------------------------------
subroutine place_elements

    use horseshoe_locations
    use geometric_inputs

    implicit none

    real (kind=8),parameter:: pi=4.*atan(1.)
    real a,b,aux(n_horseshoe+1),auxd1,auxd2
    real (kind=8):: aux_v1(n_horseshoe+1),aux_v2(n_horseshoe+1)
    integer i,j
```

```
    integer, parameter:: k=40 !number of samples taken for calculating the area of each strip
    real (kind=8)::aux_area(k),aux_points(3,k)
    auxd1=0.2
    auxd2=0.2
    a=minval(geom(1,:))
    b=maxval(geom(1,:))
    do i=1,n_horseshoe+1
        aux(i)=auxd1+(pi-auxd2-auxd1)/(n_horseshoe)*(i-1)
    end do
    do i=1,n_horseshoe+1
        !cosinus distribution of the points
        horseshoe_loc(1,i)=a+(cos(auxd1)-cos(aux(i)))/(cos(auxd1)-cos(pi-auxd2))*(b-a)
        !linear distribution of the points
        !horseshoe_loc(1,i)=a+(b-a)*(i-1)/(n_horseshoe)
    end do
    !the horseshoe nodes have been located following a cosine distribution in the spanwise direction
    horseshoe_te_loc(1,:)=horseshoe_loc(1,:)

    !cosine distribution for the control points location in spanwise direction
    do i=1,n_horseshoe
        aux(i)=auxd1+(pi-auxd2-auxd1)/(n_horseshoe)*(i-0.5)
        control_points_loc(1,i)=a+(cos(auxd1)-cos(aux(i)))/(cos(auxd1)-cos(pi-auxd2))*(b-a)
        !control_points_loc(1,i)=(horseshoe_loc(1,i)+horseshoe_loc(1,i+1))/2.
    end do

    do i=1,n_horseshoe
        call spline(geom(1,:),geom(4,:),control_points_loc(1,i),twist(i),n_point,1)
        call spline(geom(1,:),geom(5,:),control_points_loc(1,i),thickness(i),n_point,1)

    end do


    do i=2,n_horseshoe
        call spline(geom(1,:),geom(2,:),horseshoe_loc(1,i),aux_v1(i),n_point,1)
        horseshoe_te_loc(2,i)=aux_v1(i)
        call spline(geom(1,:),geom(3,:),horseshoe_loc(1,i),aux_v2(i),n_point,1)
        horseshoe_loc(2,i)=aux_v2(i)+(aux_v1(i)-aux_v2(i))*0.25

    end do
    horseshoe_loc(1,1)=a
    horseshoe_loc(1,n_horseshoe+1)=b
    horseshoe_te_loc(1,1)=a
    horseshoe_te_loc(1,n_horseshoe+1)=b
    horseshoe_te_loc(2,1)=geom(2,1)
    horseshoe_te_loc(2,n_horseshoe+1)=geom(2,n_point)
    horseshoe_loc(2,1)=geom(3,1)+(horseshoe_te_loc(2,1)-geom(3,1))*0.25
    horseshoe_loc(2,n_horseshoe+1)=geom(3,n_point)+(horseshoe_te_loc(2,n_horseshoe+1)-geom(3,n_point))*0.25
    !the horseshoe nodes have been located at the 25% of the local chord
    do i=1,n_horseshoe
        call spline(geom(1,:),geom(2,:),control_points_loc(1,i),aux_v1(i),n_point,1)
        call spline(geom(1,:),geom(3,:),control_points_loc(1,i),aux_v2(i),n_point,1)
        control_points_loc(2,i)=aux_v2(i)+(aux_v1(i)-aux_v2(i))*0.25
        call spline(geom(1,:),geom(4,:),control_points_loc(1,i),twist(i),n_point,1)
        call spline(geom(1,:),geom(5,:),control_points_loc(1,i),thickness(i),n_point,1)

        chord_distr(i)=aux_v1(i)-aux_v2(i)
    end do
    !the control points have been located at the 25% of the local chord
    !new way for calculating the area distribution
    area_distr(:)=0.d0
    do j=1,n_horseshoe
        a=horseshoe_loc(1,j)
        b=horseshoe_loc(1,j+1)
        do i=1,k
            aux_points(1,i)=a+(b-a)*(i-1)/(k-1)
            call spline(geom(1,:),geom(2,:),aux_points(1,i),aux_points(2,i),n_point,1)
            call spline(geom(1,:),geom(3,:),aux_points(1,i),aux_points(3,i),n_point,1)
        end do
        do i=1,k-1
            aux_area(i)=(aux_points(2,i)+aux_points(2,i+1)-aux_points(3,i)-aux_points(3,i+
```

```
    1))*(aux_points(1,i+1)-aux_points(1,i))/2.
            area_distr(j)=area_distr(j)+aux_area(i)
        end do

    end do

    horseshoe_loc(1,:)=horseshoe_loc(1,:)+hub_to_br
    horseshoe_te_loc(1,:)=horseshoe_te_loc(1,:)+hub_to_br
    control_points_loc(1,:)=control_points_loc(1,:)+hub_to_br

!created 02/01/2013
!validated 02/01/2013
end subroutine place_elements
!-----------------------------------
subroutine from_cartesian_to_polars
    use horseshoe_locations
    implicit none

    real(kind=8),parameter:: pi=4.*atan(1.)
    real aux1,aux2
    integer i
    allocate(hs_loc_pol(2,n_horseshoe+1))
    allocate(cp_loc_pol(2,n_horseshoe))
    do i=1,n_horseshoe
        aux1=sqrt((control_points_loc(1,i))**2)
        aux2=sqrt((control_points_loc(2,i))**2)
        cp_loc_pol(1,i)=sqrt(aux1**2+aux2**2)
        cp_loc_pol(2,i)=atan(aux2/aux1)
        if ((control_points_loc(1,i)).GE.0.) then
            if (control_points_loc(2,i).GE.0.) then
                cp_loc_pol(2,i)=cp_loc_pol(2,i)
            else
                cp_loc_pol(2,i)=2*pi-cp_loc_pol(2,i)
            end if
        else
            if (control_points_loc(2,i).GE.0.) then
                cp_loc_pol(2,i)=pi-cp_loc_pol(2,i)
            else
                cp_loc_pol(2,i)=cp_loc_pol(2,i)+pi
            end if
        end if

        aux1=sqrt((horseshoe_loc(1,i))**2)
        aux2=sqrt((horseshoe_loc(2,i))**2)
        hs_loc_pol(1,i)=sqrt(aux1**2+aux2**2)
        hs_loc_pol(2,i)=atan(aux2/aux1)
        if (horseshoe_loc(1,i).GE.0.) then
            if (horseshoe_loc(2,i).GE.0.) then
                hs_loc_pol(2,i)=hs_loc_pol(2,i)
            else
                hs_loc_pol(2,i)=2*pi-hs_loc_pol(2,i)
            end if
        else
            if (horseshoe_loc(2,i).GE.0.) then
                hs_loc_pol(2,i)=pi-hs_loc_pol(2,i)
            else
                hs_loc_pol(2,i)=hs_loc_pol(2,i)+pi
            end if
        end if

    end do
    aux1=sqrt((horseshoe_loc(1,n_horseshoe+1))**2)
    aux2=sqrt((horseshoe_loc(2,n_horseshoe+1))**2)
    hs_loc_pol(1,n_horseshoe+1)=sqrt(aux1**2+aux2**2)
    hs_loc_pol(2,n_horseshoe+1)=atan(aux2/aux1)

    if ((horseshoe_loc(1,n_horseshoe+1))/aux1.GE.0.) then
        if (horseshoe_loc(2,n_horseshoe+1).GE.0.) then
            hs_loc_pol(2,n_horseshoe+1)=hs_loc_pol(2,n_horseshoe+1)
        else
            hs_loc_pol(2,n_horseshoe+1)=hs_loc_pol(2,n_horseshoe+1)+3.*pi/2.
        end if
    else
        if (horseshoe_loc(2,n_horseshoe+1).GE.0.) then
```

```
            hs_loc_pol(2,n_horseshoe+1)=hs_loc_pol(2,n_horseshoe+1)+pi/2.
        else
            hs_loc_pol(2,n_horseshoe+1)=hs_loc_pol(2,n_horseshoe+1)+pi
        end if
    end if

!created 03/11/2012
!validated  03/11/2012 currently it is not used
end subroutine from_cartesian_to_polars
!---------------------------------------------------------------
subroutine temp_location

    use horseshoe_locations
    use aerodynamic_performance
    use clock
    use geometric_inputs
    implicit none

    integer i,b
    real(kind=8),parameter::pi=4.*atan(1.)
    real mat_az(3,3),mat_pitch(3,3),mat_c(3,3),aux3(3),aux(3),aux_a, aux_mod
    real(kind=8):: aux_c(3),aux_s(3)

    do b=1,n_blades
        !for b=1-->the variable called azimut is the corresponding azimut of the first
    blade, the other ones are evenly spaced
        aux_a=azimut+(b-1)*2*pi/n_blades

        mat_az(1,1)=1.
        mat_az(2,1)=0.
        mat_az(3,1)=0.
        mat_az(1,2)=0.
        mat_az(2,2)=-sin(aux_a)
        mat_az(3,2)=cos(aux_a)
        mat_az(1,3)=0.
        mat_az(2,3)=-cos(aux_a)
        mat_az(3,3)=-sin(aux_a)

        mat_pitch(2,1)=0.
        mat_pitch(1,2)=0.
        mat_pitch(2,2)=1.
        mat_pitch(3,2)=0.
        mat_pitch(2,3)=0.

        mat_c(1,1)=1.
        mat_c(2,1)=0.
        mat_c(3,1)=0.
        mat_c(1,2)=0.
        mat_c(2,2)=1.
        mat_c(3,2)=0.
        mat_c(1,3)=0.
        mat_c(2,3)=0.
        mat_c(3,3)=-1.



        do i=1,n_horseshoe

            mat_pitch(1,1)=cos(pitch+twist(i))
            mat_pitch(3,1)=sin(pitch+twist(i))
            mat_pitch(1,3)=-sin(pitch+twist(i))
            mat_pitch(3,3)=cos(pitch+twist(i))
            aux3(1)=0.
            aux3(2)=control_points_loc(1,i)
            aux3(3)=control_points_loc(2,i)
            aux3=matmul(mat_c,aux3)
            aux3=matmul(mat_pitch,aux3)
            temp_cp_loc(b,:,i)=matmul(mat_az,aux3)
            aux3(1)=0.
            aux3(2)=horseshoe_loc(1,i)
            aux3(3)=horseshoe_loc(2,i)
            aux3=matmul(mat_c,aux3)
            aux3=matmul(mat_pitch,aux3)
            temp_hs_loc(b,:,i)=matmul(mat_az,aux3)
```

```
            aux3(1)=0.
            aux3(2)=horseshoe_te_loc(1,i)
            aux3(3)=horseshoe_te_loc(2,i)
            aux3=matmul(mat_c,aux3)
            aux3=matmul(mat_pitch,aux3)
            temp_hs_te_loc(b,:,i)=matmul(mat_az,aux3)

        end do
        aux3(1)=0.
        aux3(2)=horseshoe_loc(1,n_horseshoe+1)
        aux3(3)=horseshoe_loc(2,n_horseshoe+1)
        aux3=matmul(mat_c,aux3)
        aux3=matmul(mat_pitch,aux3)
        temp_hs_loc(b,:,n_horseshoe+1)=matmul(mat_az,aux3)
        aux3(1)=0.
        aux3(2)=horseshoe_te_loc(1,n_horseshoe+1)
        aux3(3)=horseshoe_te_loc(2,n_horseshoe+1)
        aux3=matmul(mat_c,aux3)
        aux3=matmul(mat_pitch,aux3)
        temp_hs_te_loc(b,:,n_horseshoe+1)=matmul(mat_az,aux3)
    end do !loop for b=1,n_blades--->for calculating the positions for all blades

    do i=1,n_horseshoe
        el_vect1(time_step,1,i)=temp_hs_te_loc(1,1,i)+temp_hs_te_loc(1,1,i+1)-(temp_hs_loc↙
    (1,1,i)+temp_hs_loc(1,1,i+1))
        el_vect1(time_step,2,i)=temp_hs_te_loc(1,2,i)+temp_hs_te_loc(1,2,i+1)-(temp_hs_loc↙
    (1,2,i)+temp_hs_loc(1,2,i+1))
        el_vect1(time_step,3,i)=temp_hs_te_loc(1,3,i)+temp_hs_te_loc(1,3,i+1)-(temp_hs_loc↙
    (1,3,i)+temp_hs_loc(1,3,i+1))
        aux_mod=sqrt(el_vect1(time_step,1,i)**2+el_vect1(time_step,2,i)**2+el_vect1       ↙
    (time_step,3,i)**2)
        el_vect1(time_step,:,i)=el_vect1(time_step,:,i)/aux_mod

        aux_c(1)=temp_hs_loc(1,1,i+1)-temp_hs_loc(1,1,i)
        aux_c(2)=temp_hs_loc(1,2,i+1)-temp_hs_loc(1,2,i)
        aux_c(3)=temp_hs_loc(1,3,i+1)-temp_hs_loc(1,3,i)

        call crossprod(el_vect1(time_step,:,i),aux_c,aux_s)

        aux_mod=sqrt(aux_s(1)**2+aux_s(2)**2+aux_s(3)**2)
        el_vect3(time_step,:,i)=aux_s(:)/aux_mod

        call crossprod(el_vect3(time_step,:,i),el_vect1(time_step,:,i),el_vect2(time_step,↙
    :,i))

    end do



!created 08/11/2012
!validated 09/11/2012
!modified 12/04/2013 el_vect1,2,3 definition included
end subroutine temp_location
!---------------------------------------------------------------------
!subroutine for calculating the induced speed on control points by blade rotation
subroutine movement_ind_speed

    use horseshoe_locations
    use operating_parameters
    use incident_speeds
    use aerodynamic_performance
    implicit none

    real vind_pol(2,n_horseshoe)
    integer i

    do i=1,n_horseshoe
        vind_pol(1,i)=omega*cp_loc_pol(1,i)*sin(cp_loc_pol(2,i))
        vind_pol(2,i)=omega*cp_loc_pol(1,i)*cos(cp_loc_pol(2,i))

        turning_ind(1,i)=vind_pol(1,i)*sin(azimut)+vind_pol(2,i)*cos(azimut)
        turning_ind(2,i)=-vind_pol(1,i)*cos(azimut)+vind_pol(2,i)*sin(azimut)
    end do
!validated
```

```
end subroutine movement_ind_speed
!-------------------------------------------------------------
subroutine wind_speed
    use operating_parameters
    use horseshoe_locations
    use clock
    use incident_speeds
    use geometric_inputs
    implicit none

    integer i
    real,parameter:: pi=4*atan(1.)

    do i=1,n_horseshoe

            wind(time_step,1,i)=v0(1)*((temp_cp_loc(1,3,i)+hub_height)/hub_height)**exp_a
            wind(time_step,2,i)=v0(2)*((temp_cp_loc(1,3,i)+hub_height)/hub_height)**exp_a
            wind(time_step,3,i)=v0(3)*((temp_cp_loc(1,3,i)+hub_height)/hub_height)**exp_a

    end do

!validated 4/11/2012
!modified 17/11/2012 (dimensions of temp_hs_loc increased)
!validated if not time dependant 07/12/12
end subroutine wind_speed
!-------------------------------------------------------------
subroutine wind_speed_for_wake(locations,speeds)
    use operating_parameters
    use horseshoe_locations
    use wake_sheet
    use clock
    use simulation_data
    use geometric_inputs
    implicit none

    integer i
    real (kind=8), intent(in):: locations(3,n_horseshoe+1)
    real (kind=8), intent(out):: speeds(3,n_horseshoe+1)

    do i=1,n_horseshoe+1

        speeds(1,i)=v0(1)*((locations(3,i)+hub_height)/hub_height)**exp_a
        speeds(2,i)=v0(2)*((locations(3,i)+hub_height)/hub_height)**exp_a
        speeds(3,i)=v0(3)*((locations(3,i)+hub_height)/hub_height)**exp_a
    end do

!created 17/11/2012
!validated 07/12/12
end subroutine wind_speed_for_wake
!-------------------------------------------------------------
!subroutine that add the incident speeds on each cp at every loop iteration or time step
subroutine cp_incident_speeds

    use incident_speeds
    use aerodynamic_performance
    use horseshoe_locations
    use clock
    implicit none

    integer i,j1,j2
    real aux3(3), aux_alpha(n_horseshoe)

    do i=1,n_horseshoe
        aux3(1:3)=0.


        total_wind(time_step,1,i)=wind(time_step,1,i)+wake_ind(time_step,1,i)+blade_ind ↙
    (time_step,1,i)
        total_wind(time_step,2,i)=wind(time_step,2,i)+turning_ind(1,i)+wake_ind(time_step, ↙
    2,i)+blade_ind(time_step,2,i)
        total_wind(time_step,3,i)=wind(time_step,3,i)+turning_ind(2,i)+wake_ind(time_step, ↙
    3,i)+blade_ind(time_step,3,i)

        total_wind(time_step,4,i)=sqrt(dot_product(total_wind(time_step,1:3,i),total_wind ↙
```

```
    (time_step,1:3,i))) !all 3 components of the speed has been considered for incident
    speed module calculation
    end do
    call check
    do i=1,n_horseshoe
        alpha(time_step,i)=atan(dot_product(total_wind(time_step,1:3,i),el_vect3(time_step
    ,:,i))/dot_product(total_wind(time_step,1:3,i),el_vect1(time_step,:,i)))
    end do

    do i=1,n_horseshoe
        j1=max(1,i-2)
        j2=min(n_horseshoe,i+2)
        aux_alpha(i)=sum(alpha(time_step,j1:j2))/(j2-j1+1)
    end do

!created 01/11/2012
!modified 25/11/2012
!validated 13/01/2013
end subroutine cp_incident_speeds
!-------------------------------------------------------------
!subroutine just for giving the Cl of each strip for a given angle of attack
subroutine vortex_intensity_calculator !print lines to be removed

    use horseshoe_locations
    use aerodynamic_performance ! alpha, cl, cd cm tau, L D Q P
    use incident_speeds ! total_wind
    use airfoil_polars
    use vortex_intensities
    use iteration_data
    use clock
    implicit none
    integer i
    real (kind=8):: aux1, aux2(3),dl(3),relax,aux
    real (kind=8):: auxdot1,auxdot3,auxcros(3),denaux1,denaux3

    relax=0.1

    do i=1,n_horseshoe
        call spline(real(AOA(1,:)),real(Cl_pol(i,:)),alpha(time_step,i),Cl(time_step,i),
    n_data_points,1)
    end do

    open(UNIT=6,FILE='Test_aux3_VorIntCal.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
    do i=1,n_horseshoe
        aux_attached_hs_int(i)=pr_attached_hs_int(time_step,i)

        dl(1)=temp_hs_loc(1,1,i+1)-temp_hs_loc(1,1,i)
        dl(2)=temp_hs_loc(1,2,i+1)-temp_hs_loc(1,2,i)
        dl(3)=temp_hs_loc(1,3,i+1)-temp_hs_loc(1,3,i)

        auxdot1=dot_product(total_wind(time_step,1:3,i),el_vect1(time_step,:,i))
        auxdot3=dot_product(total_wind(time_step,1:3,i),el_vect3(time_step,:,i))

        call crossprod(total_wind(time_step,1:3,i),dl,auxcros)

        denaux1=dot_product(auxcros,el_vect1(time_step,:,i))
        denaux3=dot_product(auxcros,el_vect3(time_step,:,i))

        pr_attached_hs_int(time_step,i)=(1.-relax)*pr_attached_hs_int(time_step,i)+relax*
    (0.5*Cl(time_step,i)*(auxdot1**2+auxdot3**2)*area_distr(i)/sqrt(denaux1**2+denaux3**
    2))

        call crossprod(total_wind(time_step,1:3,i),dl,aux2)

        aux1=sqrt(dot_product(aux2,aux2))

        write(6,*) pr_attached_hs_int(time_step,i)


    end do
    write(6,*) 'End of file'
    close (6)
!created 15/11/12
!validated 13/01/13
```

```
end subroutine vortex_intensity_calculator
!---------------------------------------------------------------
!subroutine only with validation purposes, it will save some variables for test the code
    in different conditions before its release
subroutine check

    use horseshoe_locations
    use operating_parameters
    use incident_speeds
    use vortex_intensities
    use wake_sheet
    use aerodynamic_performance
    use tf_matrix
    use clock
    use simulation_data
    use airfoil_polars
    implicit none

    integer i,j,b_vortex_strength,b_wake_pos,b_hs_pos, b_cp_pos,b_AOA_dist,b_wake_int,
    b_tf_mat,b_blade_ind_speed,b_wake_ind_speed,b_cp_inc_speeds,b_cl,b_wind_speed,
    b_area_dist,b_chord_dist,b_hs_cp_2D_pos, b_wake_connec,b_sec_pol
    real (kind=8)::aux
    b_cp_pos=0
    b_hs_pos=0
    b_wake_pos=0
    b_vortex_strength=0
    b_AOA_dist=0
    b_wake_int=0
    b_tf_mat=0
    b_blade_ind_speed=0
    b_wake_ind_speed=0
    b_cp_inc_speeds=0
    b_wake_connec=0
    b_cl=0
    b_wind_speed=0
    b_area_dist=0
    b_chord_dist=0
    b_hs_cp_2D_pos=0
    b_sec_pol=0

    if (b_cp_pos.NE.0) then
        open(UNIT=6,FILE='Control_points_pos.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        write(6,*) 'Number of control points:', n_horseshoe
        write(6,*) 'Control points locations'
        write(6,*) 'x  y  z'
        do i=1,n_horseshoe
            write(6,*) temp_cp_loc(:,:,i)
        end do
        write(6,*) 'End of file'
        close (6)
    end if
    if (b_hs_pos.NE.0) then
        open(UNIT=6,FILE='Horseshoe_nodes_pos.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        write(6,*) 'Number of horseshoe nodes:', n_horseshoe
        write(6,*) 'Horseshoe locations'
        write(6,*) 'x  y  z'
        do i=1,n_horseshoe+1
            write(6,*) temp_hs_loc(1,:,i)
        end do
        write(6,*) 'End of file'
        close (6)
    end if
    if (b_vortex_strength.NE.0) then
        open(UNIT=6,FILE='Attached_vortex_strength_distribution.txt',ACCESS='SEQUENTIAL',
    STATUS='REPLACE')
        write(6,*) 'Number of segments:', n_horseshoe
        write(6,*) 'Segments intensity'
        do j=1,n_time_steps
            do i=1,n_horseshoe
                write(6,*) pr_attached_hs_int(j,i)
            end do
        end do
        write(6,*) 'End of file'
        close (6)
```

```
    end if
    if (b_AOA_dist.NE.0) then
        open(UNIT=6,FILE='Angle_of_attack_distribution.txt',ACCESS='SEQUENTIAL',STATUS=
    'REPLACE')
        write(6,*) 'Number of control points:', n_horseshoe
        write(6,*) 'Angle of attack at each control point [rad]'
        do j=1,n_time_steps
            do i=1,n_horseshoe
                write(6,*) alpha(j,i)
            end do
        end do
        write(6,*) 'End of file'
        close (6)
    end if
    if (b_wake_pos.NE.0) then
        open(UNIT=6,FILE='Wake_nodes_pos.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        write(6,*) 'Total number of nodes:', n_horseshoe*n_wake_lines
        write(6,*) 'Number of lines of nodes', n_wake_lines
        write(6,*) 'x  y  z'
        do i=1,(n_horseshoe+1)*n_wake_lines
                !write(6,*) pr_wake_nodes_loc(blade_phase(1,time_step),:,j+(i-1)*
    (n_horseshoe+1))
                write(6,*) pr_wake_nodes_loc(1,:,i)
        end do
        write(6,*) 'End of file'
        close (6)
    end if
    if (b_wake_int.NE.0) then
        open(UNIT=6,FILE='Wake_vortex_int.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        write(6,*) 'Total number of shed vortex:', n_horseshoe
        write(6,*) 'Intensities'
        do i=1,n_horseshoe*(n_wake_lines-1)
            !write(6,*) pr_wake_int(blade_phase(1,time_step),i)
            write(6,*) pr_wake_int(:,i)
        end do
        write(6,*) 'End of file'
        close (6)
    end if
    if (b_tf_mat.NE.0) then
        open(UNIT=6,FILE='Attached_s_matrix_u.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_horseshoe
            do j=1,n_horseshoe
                write(6,*) attached_s_matrix_u(i,j)
            end do
        end do
        write(6,*) 'End of document'
        close (6)
        open(UNIT=6,FILE='Attached_s_matrix_v.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_horseshoe
            do j=1,n_horseshoe
                write(6,*) attached_s_matrix_v(i,j)
            end do
        end do
        write(6,*) 'End of document'
        close (6)
        open(UNIT=6,FILE='Attached_s_matrix_w.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_horseshoe
            do j=1,n_horseshoe
                write(6,*) attached_s_matrix_w(i,j)
            end do
        end do
        write(6,*) 'End of document'
        close (6)
        open(UNIT=6,FILE='Attached_c_matrix_u.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_horseshoe
            do j=1,n_horseshoe+1
                write(6,*) attached_c_matrix_u(i,j)
            end do
        end do
        write(6,*) 'End of document'
        close (6)
        open(UNIT=6,FILE='Attached_c_matrix_v.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_horseshoe
            do j=1,n_horseshoe+1
```

```
                write(6,*) attached_c_matrix_v(i,j)
            end do
        end do
        write(6,*) 'End of document'
        close (6)
        open(UNIT=6,FILE='Attached_c_matrix_w.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_horseshoe
            do j=1,n_horseshoe+1
                write(6,*) attached_c_matrix_w(i,j)
            end do
        end do
        write(6,*) 'End of document'
        close (6)
    end if
    if (b_blade_ind_speed.NE.0) then
        open(UNIT=6,FILE='Blade_induced_speeds_u.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE'
    )
        do i=1,n_horseshoe
            write(6,*) blade_ind(1,:,i)
        end do
        write(6,*) 'End of document'
        close (6)
    end if
    if (b_wake_ind_speed.NE.0) then
        open(UNIT=6,FILE='Wake_induced_speeds.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_horseshoe
            write(6,*) wake_ind(1,:,i)
        end do
        write(6,*) 'End of document'
        close (6)
        open(UNIT=6,FILE='Wake2_induced_speeds.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_horseshoe
            write(6,*) wake_ind2(1,:,i)
        end do
        write(6,*) 'End of document'
        close (6)
    end if
    if (b_cp_inc_speeds.NE.0) then
        open(UNIT=6,FILE='CP_incident_speeds.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_horseshoe
            write(6,*) total_wind(1,:,i)
        end do
        write(6,*) 'End of document'
        close (6)
    end if
    if (b_cl.NE.0) then
        open(UNIT=6,FILE='Cl_distribution.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do j=1,n_time_steps
            do i=1,n_horseshoe
                write(6,*) Cl(j,i)
            end do
        end do
        write(6,*) 'End of document'
        close (6)
    end if
    if (b_wind_speed.NE.0) then
        open(UNIT=6,FILE='Wind_speed_incidence.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_horseshoe
            write(6,*) wind(1,:,i)
        end do
        write(6,*) 'End of document'
        close (6)
    end if
    if (b_area_dist.NE.0) then
        open(UNIT=6,FILE='Area_per_element.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_horseshoe
            write(6,*) area_distr(i)
        end do
        write(6,*) 'End of document'
        close (6)
    end if
    if (b_chord_dist.NE.0) then
        open(UNIT=6,FILE='Chord_distribution.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_horseshoe
```

```
            write(6,*) chord_distr(i)
        end do
        write(6,*) 'End of document'
        close (6)
    end if
    if (b_hs_cp_2D_pos.NE.0) then
        open(UNIT=6,FILE='CP_2D_locations.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        write(6,*) 'Number of control points:', n_horseshoe
        write(6,*) 'Control points locations'
        write(6,*) 'x  y  z'
        do i=1,n_horseshoe
            write(6,*) control_points_loc(:,i)
        end do
        write(6,*) 'End of file'
        close (6)
        open(UNIT=6,FILE='HS_2D_locations.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        write(6,*) 'Number of horseshoe nodes:', n_horseshoe+1
        write(6,*) 'HS nodes locations'
        write(6,*) 'x  y  z'
        do i=1,n_horseshoe+1
            write(6,*) horseshoe_loc(:,i)
        end do
        write(6,*) 'End of file'
        close (6)
        open(UNIT=6,FILE='HS_te_2D_locations.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        write(6,*) 'Number of horseshoe te nodes:', n_horseshoe+1
        write(6,*) 'HS te nodes locations'
        write(6,*) 'x  y  z'
        do i=1,n_horseshoe+1
            write(6,*) horseshoe_te_loc(:,i)
        end do
        write(6,*) 'End of file'
        close (6)
    end if
    if (b_wake_connec.NE.0) then
        open(UNIT=6,FILE='Wake_nodes_connectivity.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE
    ')
        do i=1,(n_wake_lines-1)*(n_horseshoe+1)
            write(6,*) wake_nodes_connec(:,i)
        end do
        write(6,*) 'End of file'
        close (6)
    end if
    if (b_sec_pol.NE.0) then
        open(UNIT=6,FILE='Section_polars.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_horseshoe
            write(6,*) "new section"
            do j=1,n_data_points
                write(6,*) Cl_pol(i,j)
            end do
        end do
        write(6,*) "end of file"
        close(6)
    end if
    open(UNIT=6,FILE='Turning_induced_speed.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
    do i=1,n_horseshoe
        write(6,*) turning_ind(:,i)
    end do
    close(6)

!Further writing codes can be add easily

!created 18/11/2012
!validated 07/12/12
end subroutine check
!-------------------------------------------------------------
!subroutine for calculating the wake rings vortex intensities, from the vortex strength
    distribution along the blade
!this subroutine is just intended for stationary cases, since the wake strength is
    constant for each spanwise position
!it will be used aslo for the first approximation when building the wake
!for transient conditions it is necessary to use another subroutine
subroutine wake_vortex_intensities
    use horseshoe_locations
```

```
    use wake_sheet
    use vortex_intensities
    use clock
    use iteration_data
    use simulation_data
    implicit none

    integer i,j,k

    if (wake_int_init.EQ.1) then
        k=1
        do i=1,n_horseshoe
            do j=1,n_wake_lines-1
                pr_wake_int(:,(j-1)*n_horseshoe+i)=pr_attached_hs_int(blade_phase(1,
    time_step+1),i)
            end do
        end do

    else if (wake_int_init.EQ.2) then

        print*, "time(time_step) ", time(time_step)

        if (time_sample.NE.time(time_step)) then

            do i=1,n_horseshoe*(n_wake_lines-2)
                pr_wake_int(blade_phase(1,time_step+1),i+n_horseshoe)=pr_wake_int
    (blade_phase(1,time_step),i)
            end do
            do i=1,n_horseshoe
                pr_wake_int(time_step,i)=pr_attached_hs_int(blade_phase(1,time_step),i)
            end do
            time_sample=time(time_step)
        end if
    else
        print*,"There was an error. The value of wake_int_init must be contained within 1
    and 2"
        pause
        stop
    end if
!created 18/01/2013
!validated
end subroutine wake_vortex_intensities
!-------------------------------------------------------
!subroutine for calculating the speed induced by the wake
subroutine wake_induced_speed

    use horseshoe_locations
    use incident_speeds
    use vortex_intensities
    use wake_sheet
    use clock
    use geometric_inputs
    use simulation_data
    implicit none

    integer i,j,k,b,q
    real aux3(3)
    real (kind=8):: auxpos(2,3),speeds(3)


    do i=1,n_horseshoe
        aux3(1:3)=0.
        !test
        k=1
        q=0
        do b=1,n_blades
            do j=1,(n_wake_lines-1)*n_horseshoe

                auxpos(1,1)=pr_wake_nodes_loc(blade_phase(b,time_step+1),1,
    wake_nodes_connec(2,j))
                auxpos(1,2)=pr_wake_nodes_loc(blade_phase(b,time_step+1),2,
    wake_nodes_connec(2,j))
                auxpos(1,3)=pr_wake_nodes_loc(blade_phase(b,time_step+1),3,
    wake_nodes_connec(2,j))
```

```
                auxpos(2,1)=pr_wake_nodes_loc(blade_phase(b,time_step+1),1,
    wake_nodes_connec(3,j))
                auxpos(2,2)=pr_wake_nodes_loc(blade_phase(b,time_step+1),2,
    wake_nodes_connec(3,j))
                auxpos(2,3)=pr_wake_nodes_loc(blade_phase(b,time_step+1),3,
    wake_nodes_connec(3,j))
            !auxpos is the position of the nodes that delimits the vortex segment
                call VLtfnc(temp_cp_loc(1,1,i),temp_cp_loc(1,2,i),temp_cp_loc(1,3,i),
    auxpos(1,1),auxpos(1,2),auxpos(1,3),auxpos(2,1),auxpos(2,2),auxpos(2,3),pr_wake_int
    (blade_phase(b,time_step+1),wake_nodes_connec(1,j)),speeds(1),speeds(2),speeds(3))
                !test
                !call VLtfnc(temp_cp_loc(1,1,i),temp_cp_loc(1,2,i),temp_cp_loc(1,3,i),
    auxpos(1,1),auxpos(1,2),auxpos(1,3),auxpos(2,1),auxpos(2,2),auxpos(2,3),1.d0,speeds(1)
    ,speeds(2),speeds(3))

                !if (j.GT.n_horseshoe) then
                    aux3(1)=aux3(1)+speeds(1)
                    aux3(2)=aux3(2)+speeds(2)
                    aux3(3)=aux3(3)+speeds(3)
                !end if

                auxpos(1,1)=pr_wake_nodes_loc(blade_phase(b,time_step+1),1,
    wake_nodes_connec(3,j))
                auxpos(1,2)=pr_wake_nodes_loc(blade_phase(b,time_step+1),2,
    wake_nodes_connec(3,j))
                auxpos(1,3)=pr_wake_nodes_loc(blade_phase(b,time_step+1),3,
    wake_nodes_connec(3,j))
                auxpos(2,1)=pr_wake_nodes_loc(blade_phase(b,time_step+1),1,
    wake_nodes_connec(5,j))
                auxpos(2,2)=pr_wake_nodes_loc(blade_phase(b,time_step+1),2,
    wake_nodes_connec(5,j))
                auxpos(2,3)=pr_wake_nodes_loc(blade_phase(b,time_step+1),3,
    wake_nodes_connec(5,j))
            !auxpos is the position of the nodes that delimits the vortex segment
                call VLtfnc(temp_cp_loc(1,1,i),temp_cp_loc(1,2,i),temp_cp_loc(1,3,i),
    auxpos(1,1),auxpos(1,2),auxpos(1,3),auxpos(2,1),auxpos(2,2),auxpos(2,3),pr_wake_int
    (blade_phase(b,time_step+1),wake_nodes_connec(1,j)),speeds(1),speeds(2),speeds(3))
                !test
                !call VLtfnc(temp_cp_loc(1,1,i),temp_cp_loc(1,2,i),temp_cp_loc(1,3,i),
    auxpos(1,1),auxpos(1,2),auxpos(1,3),auxpos(2,1),auxpos(2,2),auxpos(2,3),1.d0,speeds(1)
    ,speeds(2),speeds(3))
                !if (j.EQ.n_horseshoe*k) then
                !   k=k+1

                    aux3(1)=aux3(1)+speeds(1)
                    aux3(2)=aux3(2)+speeds(2)
                    aux3(3)=aux3(3)+speeds(3)
                !end if

                auxpos(1,1)=pr_wake_nodes_loc(blade_phase(b,time_step+1),1,
    wake_nodes_connec(5,j))
                auxpos(1,2)=pr_wake_nodes_loc(blade_phase(b,time_step+1),2,
    wake_nodes_connec(5,j))
                auxpos(1,3)=pr_wake_nodes_loc(blade_phase(b,time_step+1),3,
    wake_nodes_connec(5,j))
                auxpos(2,1)=pr_wake_nodes_loc(blade_phase(b,time_step+1),1,
    wake_nodes_connec(4,j))
                auxpos(2,2)=pr_wake_nodes_loc(blade_phase(b,time_step+1),2,
    wake_nodes_connec(4,j))
                auxpos(2,3)=pr_wake_nodes_loc(blade_phase(b,time_step+1),3,
    wake_nodes_connec(4,j))
            !auxpos is the position of the nodes that delimits the vortex segment
                call VLtfnc(temp_cp_loc(1,1,i),temp_cp_loc(1,2,i),temp_cp_loc(1,3,i),
    auxpos(1,1),auxpos(1,2),auxpos(1,3),auxpos(2,1),auxpos(2,2),auxpos(2,3),pr_wake_int
    (blade_phase(b,time_step+1),wake_nodes_connec(1,j)),speeds(1),speeds(2),speeds(3))
                !test
                !call VLtfnc(temp_cp_loc(1,1,i),temp_cp_loc(1,2,i),temp_cp_loc(1,3,i),
    auxpos(1,1),auxpos(1,2),auxpos(1,3),auxpos(2,1),auxpos(2,2),auxpos(2,3),1.d0,speeds(1)
    ,speeds(2),speeds(3))
                aux3(1)=aux3(1)+speeds(1)
                aux3(2)=aux3(2)+speeds(2)
                aux3(3)=aux3(3)+speeds(3)

                auxpos(1,1)=pr_wake_nodes_loc(blade_phase(b,time_step+1),1,
```

```
    wake_nodes_connec(4,j))
                auxpos(1,2)=pr_wake_nodes_loc(blade_phase(b,time_step+1),2,
    wake_nodes_connec(4,j))
                auxpos(1,3)=pr_wake_nodes_loc(blade_phase(b,time_step+1),3,
    wake_nodes_connec(4,j))
                auxpos(2,1)=pr_wake_nodes_loc(blade_phase(b,time_step+1),1,
    wake_nodes_connec(2,j))
                auxpos(2,2)=pr_wake_nodes_loc(blade_phase(b,time_step+1),2,
    wake_nodes_connec(2,j))
                auxpos(2,3)=pr_wake_nodes_loc(blade_phase(b,time_step+1),3,
    wake_nodes_connec(2,j))
            !auxpos is the position of the nodes that delimits the vortex segment
                call VLtfnc(temp_cp_loc(1,1,i),temp_cp_loc(1,2,i),temp_cp_loc(1,3,i),
    auxpos(1,1),auxpos(1,2),auxpos(1,3),auxpos(2,1),auxpos(2,2),auxpos(2,3),pr_wake_int
    (blade_phase(b,time_step+1),wake_nodes_connec(1,j)),speeds(1),speeds(2),speeds(3))
                !test
                !call VLtfnc(temp_cp_loc(1,1,i),temp_cp_loc(1,2,i),temp_cp_loc(1,3,i),
    auxpos(1,1),auxpos(1,2),auxpos(1,3),auxpos(2,1),auxpos(2,2),auxpos(2,3),1.d0,speeds(1)
    ,speeds(2),speeds(3))
                !if (j-1.EQ.q*n_horseshoe) then
                    !q=q+1
                    aux3(1)=aux3(1)+speeds(1)
                    aux3(2)=aux3(2)+speeds(2)
                    aux3(3)=aux3(3)+speeds(3)
                !end if

            end do !loop for j
        end do !loop for b
        wake_ind(time_step,1,i)=aux3(1)
        wake_ind(time_step,2,i)=aux3(2)
        wake_ind(time_step,3,i)=aux3(3)
    end do

!created 28/11/2012
!validated 06/12/12
end subroutine wake_induced_speed

!---------------------------------------------------------------------
!subroutine for calling the other subroutines in the proper order with the objective of
    the final convergence of the vortex intensities
subroutine calculations

    use iteration_data
    use vortex_intensities
    use horseshoe_locations
    use clock
    use operating_parameters
    use aerodynamic_performance
    use simulation_data

    !test
    use operating_parameters
    implicit none

    integer i,j,k,m
    real convergence_t,convergence_i,precision_t,precision_i,aux
    real (kind=8), parameter:: pi=4*atan(1.)

    precision_t=0.01
    precision_i=0.0001

    pr_attached_hs_int(:,:)=0.d0
    pr_cycle_att_hs_int(:,:)=1.d0
!
    j=0
    m=0
    convergence_t=100
    do while (convergence_t.GT.precision_t)
        j=j+1
        print*,"J",j
        do time_step=1,n_time_steps

            convergence_i=100
            azimut=preazimut+(time_step-1)*omega*inc_time
```

```
        if ((b_stationary_sim.EQ.1).AND.(b_wake_roll_up.EQ.1)) then
            m=m+1
            azimut=preazimut+(m-1)*omega*inc_time
        end if

        call temp_location
        do i=1,n_horseshoe
            cp_saved_pos(time_step,:,i)=temp_cp_loc(1,:,i)
        end do

        call movement_ind_speed

        call wind_speed

        call place_wake_sheet

        if (b_stationary_sim.NE.1) then

            call wake_vortex_intensities

            call wake_induced_speed
        end if


        k=0
        do while (convergence_i.GT.precision_i)

            k=k+1
            print*, "K", k

            convergence_i=0
            if (b_stationary_sim.EQ.1) then

                call wake_vortex_intensities

                call wake_induced_speed
            end if

            call blade_direct_induced_speed

            call cp_incident_speeds
            call check
            call vortex_intensity_calculator

            do i=1,n_horseshoe
                aux=abs((pr_attached_hs_int(time_step,i)-aux_attached_hs_int(i))/(abs ↙
(pr_attached_hs_int(time_step,i))+0.1))
                aux_attached_hs_int(i)=pr_attached_hs_int(time_step,i)
                if (aux.GT.convergence_i) then
                    convergence_i=aux
                end if
            end do
            print*,"convergence_i=",convergence_i
            print*,"sum of Cl",sum(Cl(time_step,:))
            call check
        end do
        if (wake_int_init.EQ.1.AND.b_stationary_sim.EQ.0) then
            wake_int_init=2
        end if

    end do

    convergence_t=0
    do time_step=1,n_time_steps
        do i=1,n_horseshoe
            aux=abs(pr_attached_hs_int(time_step,i)-pr_cycle_att_hs_int(time_step,i))/↙
(abs(pr_attached_hs_int(time_step,i))+0.1)
            pr_cycle_att_hs_int(time_step,i)=pr_attached_hs_int(time_step,i)
            if (aux.GT.convergence_t) then
                convergence_t=aux
            end if
        end do
    end do
```

```
        print*,"convergence_t=",convergence_t
    end do
    print*,"Simulation finalized"

end subroutine calculations
!---------------------------------------------------------------------
!subroutine for initializing all the needed variables, to be used after reading the inputs
subroutine initialize

    use horseshoe_locations
    use aerodynamic_performance
    use airfoil_polars
    use geometric_inputs
    use incident_speeds
    use vortex_intensities
    use wake_sheet
    use tf_matrix
    use iteration_data
    use operating_parameters
    use simulation_data
    use clock
    implicit none

    real (kind=8),parameter::pi=4.*atan(1.)
    integer i, aux

    allocate(horseshoe_loc(2,n_horseshoe+1))
    allocate(horseshoe_te_loc(2,n_horseshoe+1))
    allocate(control_points_loc(2,n_horseshoe))
    allocate(area_distr(n_horseshoe))
    allocate(chord_distr(n_horseshoe))
    allocate(twist(n_horseshoe))
    allocate(thickness(n_horseshoe))
    allocate(temp_hs_loc(n_blades,3,n_horseshoe+1))
    allocate(temp_hs_te_loc(n_blades,3,n_horseshoe+1))
    allocate(temp_cp_loc(n_blades,3,n_horseshoe))
    allocate(cp_saved_pos(n_time_steps,3,n_horseshoe))
    allocate(wake_ind(n_time_steps,3,n_horseshoe))

    allocate(el_vect1(n_time_steps,3,n_horseshoe))
    allocate(el_vect2(n_time_steps,3,n_horseshoe))
    allocate(el_vect3(n_time_steps,3,n_horseshoe))

    allocate(wind(n_time_steps,3,n_horseshoe))
    allocate(turning_ind(2,n_horseshoe))
    allocate(blade_ind(n_time_steps,3,n_horseshoe))
    allocate(blade_ind2(3,n_horseshoe))
    allocate(total_wind(n_time_steps,4,n_horseshoe))
    allocate(alpha(n_time_steps,n_horseshoe))
    allocate(Cl(n_time_steps,n_horseshoe))
    allocate(Cd(n_time_steps,n_horseshoe))
    allocate(Cm(n_time_steps,n_horseshoe))
    allocate(L(n_time_steps,n_horseshoe))
    allocate(D(n_time_steps,n_horseshoe))
    allocate(Mz(n_time_steps,n_horseshoe))
    allocate(MB_ip(n_time_steps,n_horseshoe))
    allocate(MB_op(n_time_steps,n_horseshoe))
    allocate(Q(n_time_steps))
    allocate(T(n_time_steps))
    allocate(P(n_time_steps))
    allocate(Cp(n_time_steps))
    allocate(Ct(n_time_steps))
    allocate(attached_s_matrix_u(n_horseshoe,n_horseshoe))
    allocate(attached_s_matrix_v(n_horseshoe,n_horseshoe))
    allocate(attached_s_matrix_w(n_horseshoe,n_horseshoe))
    allocate(attached_c_matrix_u(n_horseshoe,n_horseshoe+1))
    allocate(attached_c_matrix_v(n_horseshoe,n_horseshoe+1))
    allocate(attached_c_matrix_w(n_horseshoe,n_horseshoe+1))
    aux=n_horseshoe*n_wake_lines

    allocate(wake_s_matrix_u(n_blades,n_horseshoe,aux))
    allocate(wake_s_matrix_v(n_blades,n_horseshoe,aux))
    allocate(wake_s_matrix_w(n_blades,n_horseshoe,aux))
    allocate(wake_segments_int_s(n_blades,aux))
```

```
    aux=(n_horseshoe+1)*(n_wake_lines-1)

    allocate(wake_c_matrix_u(n_blades,n_horseshoe,aux))
    allocate(wake_c_matrix_v(n_blades,n_horseshoe,aux))
    allocate(wake_c_matrix_w(n_blades,n_horseshoe,aux))
    allocate(wake_segments_int_c(n_blades,aux))
    print*,n_horseshoe,n_wake_lines

    allocate(wake_nodes_connec(5,n_horseshoe*(n_wake_lines-1)))
    allocate(pr_attached_hs_int(n_time_steps,n_horseshoe))
    allocate(pr_wake_int(n_time_steps,n_horseshoe*(n_wake_lines-1)))
    allocate(pr_wake_nodes_loc(n_time_steps,3,(n_horseshoe+1)*n_wake_lines))
    allocate(time(n_time_steps))
    allocate(aux_attached_hs_int(n_horseshoe))
    allocate(pr_cycle_att_hs_int(n_time_steps,n_horseshoe))

    if (omega.EQ.0) then
        inc_time=0.1
    else if (n_time_steps.GT.1) then
        inc_time=2*pi/(omega*n_time_steps)
    else
        inc_time=0.1/omega
    end if
    do i=1,n_time_steps
        time(i)=inc_time*(i-1)
    end do
    time_sample=time(n_time_steps)



!created 08/12/12
!validated 09/12/12
end subroutine initialize
!-----------------------------------------------------------------------------
!subroutine for constructing the matrix that contains the temporal relations between the ↙
    blades of the rotor and the previous positions
subroutine phase_between_blades

    use geometric_inputs
    use clock
    use simulation_data
    use horseshoe_locations
    implicit none

    integer i,b,aux(2*n_time_steps)

    do i=1,n_time_steps
        aux(i)=i
        aux(i+n_time_steps)=i
    end do

    allocate(blade_phase(n_blades,n_time_steps+1))

    do i=1,n_time_steps+1
        blade_phase(1,i)=aux(n_time_steps+i-1)
        do b=2,n_blades
            blade_phase(b,i)=aux(i+(b-1)*int(n_time_steps/n_blades)-1)
        end do
    end do
    print*, "Blade Phase", blade_phase

end subroutine phase_between_blades
!-----------------------------------------------------------------------------
!subroutine for interpolating the polar curves for each defined section
subroutine extrude_polars

    use horseshoe_locations
    use airfoil_polars

    implicit none

    integer i,j
    do i=1,n_horseshoe
        do j=1,n_data_points
```

```
            call spline(airfoils_thickness,real(Cl_pol_aux(:,j)),thickness(i),Cl_pol(i,j),
    n_airfoils,1)
            call spline(airfoils_thickness,real(Cd_pol_aux(:,j)),thickness(i),Cd_pol(i,j),
    n_airfoils,1)
            call spline(airfoils_thickness,real(Cm_pol_aux(:,j)),thickness(i),Cm_pol(i,j),
    n_airfoils,1)
        end do
    end do
    open(UNIT=7,FILE='Results_polar_curves_sections.txt',ACCESS='SEQUENTIAL',STATUS=
    'REPLACE')
        do i=1,n_horseshoe
            do j=1,n_data_points
                write(7,*) Cl_pol(i,j), Cd_pol(i,j), Cm_pol(i,j)
            end do
        end do
    close(7)
!created 24/03/2013
!not validated
end subroutine extrude_polars
!---------------------------------------------------------------------
!subroutine for creating and filling the .txt files with the results of the simulations
subroutine write_outputs

    use horseshoe_locations
    use geometric_inputs
    use clock
    use operating_parameters
    use aerodynamic_performance
    use incident_speeds
    use wake_sheet
    use simulation_data
    use vortex_intensities
    use airfoil_polars

    implicit none

    integer i,j
    real, parameter:: pi=4*atan(1.)

    !lift coefficient cl, drag coefficient cd and aerodynamic moment cm
    open(UNIT=7,FILE='Results_aerodynamic_coefficients.txt',ACCESS='SEQUENTIAL',STATUS=
    'REPLACE')
    do i=1,n_time_steps
        do j=1,n_horseshoe
            write(7,*) Cl(i,j) , Cd(i,j), Cm(i,j)
        end do
    end do
    close(7)

    !angles of attack alpha
    open(UNIT=7,FILE='Results_AoA.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
    do i=1,n_time_steps
        do j=1,n_horseshoe
            write(7,*) alpha(i,j)
        end do
    end do
    close(7)

    !position of the wake nodes
    open(UNIT=7,FILE='Results_wake_nodes_pos.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
    do i=1,n_time_steps
        do j=1,(n_horseshoe+1)*n_wake_lines
            write(7,*) pr_wake_nodes_loc(i,1,j), pr_wake_nodes_loc(i,2,j),
    pr_wake_nodes_loc(i,3,j)
        end do
    end do
    close(7)
    !wake rings intensity
    open(UNIT=7,FILE='Results_wake_rings_intensity.txt',ACCESS='SEQUENTIAL',STATUS=
    'REPLACE')
    do i=1,n_time_steps
        do j=1,n_horseshoe*(n_wake_lines-1)
            write(7,*) pr_wake_int(i,j)
        end do
```

```
    end do

    !some of the inputs of the calculation
    open(UNIT=7,FILE='Results_Calculation_inputs.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE'
    )
        write(7,*) n_horseshoe
        write(7,*) n_wake_lines
        write(7,*) n_time_steps
        write(7,*) omega
        write(7,*) v0(1)
        write(7,*) v0(2)
        write(7,*) v0(3)
        write(7,*) density
        write(7,*) pitch
        write(7,*) inc_time

    close(7)

    !position of the control points and the horseshoe nodes
    open(UNIT=7,FILE='Results_cp_and_hs_positions.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE
    ')
        do i=1,n_horseshoe
            write(7,*) temp_cp_loc(1,1,i), temp_cp_loc(1,2,i), temp_cp_loc(1,2,i)
        end do
        do i=1,n_horseshoe+1
            write(7,*) temp_hs_loc(1,1,i), temp_hs_loc(1,2,i), temp_hs_loc(1,3,i)
        end do
        do i=1,n_horseshoe+1
            write(7,*) temp_hs_te_loc(1,1,i), temp_hs_te_loc(1,2,i), temp_hs_te_loc(1,3,i)
        end do
    close(7)

    !2D position of the control points and the horseshoe nodes
    open(UNIT=7,FILE='Results_2D_cp_and_hs_positions.txt',ACCESS='SEQUENTIAL',STATUS=
    'REPLACE')
        do i=1,n_horseshoe
            write(7,*) control_points_loc(1,i), control_points_loc(2,i)
        end do
        do i=1,n_horseshoe+1
            write(7,*) horseshoe_loc(1,i),horseshoe_loc(2,i)
        end do
        do i=1,n_horseshoe+1
            write(7,*) horseshoe_te_loc(1,i),horseshoe_te_loc(2,i)
        end do
    close(7)
    !geometrical properties along the blade
    open(UNIT=7,FILE='Results_geometrical_blade_properties.txt',ACCESS='SEQUENTIAL',STATUS
    ='REPLACE')
        do i=1,n_horseshoe
            write(7,*) chord_distr(i), thickness(i), twist(i)*180/pi, area_distr(i)
        end do

    close(7)
    !attached horseshoe intensity
    open(UNIT=7,FILE='Results_attached_hs_vortex_intensity.txt',ACCESS='SEQUENTIAL',STATUS
    ='REPLACE')
        do i=1,n_time_steps
            do j=1,n_horseshoe
                write(7,*) pr_attached_hs_int(i,j)
            end do
        end do
    close(7)
    !element vectors
    open(UNIT=7,FILE='Results_local_element_vectors.txt',ACCESS='SEQUENTIAL',STATUS=
    'REPLACE')
        do i=1,n_time_steps
            do j=1,n_horseshoe
                write(7,*) el_vect1(i,:,j)
            end do
        end do
        do i=1,n_time_steps
            do j=1,n_horseshoe
                write(7,*) el_vect2(i,:,j)
            end do
```

```
        end do
        do i=1,n_time_steps
            do j=1,n_horseshoe
                write(7,*) el_vect3(i,:,j)
            end do
        end do
    close(7)
    !blade and wake induced speeds
    open(UNIT=7,FILE='Results_blade_ind_speeds.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do j=1,n_time_steps
            do i=1,n_horseshoe
                write(7,*) blade_ind(j,1,i),blade_ind(j,2,i),blade_ind(j,3,i)
            end do
        end do
    close(7)
    open(UNIT=7,FILE='Results_wake_ind_speeds.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do j=1,n_time_steps
            do i=1,n_horseshoe
                write(7,*) wake_ind(j,1,i),wake_ind(j,2,i),wake_ind(j,3,i)
            end do
        end do
    close(7)
    !wind speed
    open(UNIT=7,FILE='Results_unaffected_wind_speed.txt',ACCESS='SEQUENTIAL',STATUS=
    'REPLACE')
        do i=1,n_time_steps
            do j=1,n_horseshoe
                write(7,*) wind(i,1,j), wind(i,2,j),wind(i,3,j)
            end do
        end do
    close(7)
    !total wind incidence
    open(UNIT=7,FILE='Results_incident_speeds.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_time_steps
            do j=1,n_horseshoe
                write(7,*) total_wind(i,1,j), total_wind(i,2,j), total_wind(i,3,j)
            end do
        end do
    close(7)
    !Forces and moments for the rotor
    open(UNIT=7,FILE='Results_blade_dynamics.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_time_steps
            do j=1,n_horseshoe
                write(7,*) L(i,j), D(i,j), Mz(i,j)
            end do
        end do
    close(7)
    open(UNIT=7,FILE='Results_rotor_dynamics.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_time_steps
            write(7,*) Q(i),T(i),P(i)
        end do
    close(7)
    !Performance coeficients
    open(UNIT=7,FILE='Results_rotor_coefficients.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE'
    )
        do i=1,n_time_steps
            write(7,*) Cp(i),Ct(i)
        end do
    close(7)
    !Bending moments in blades
    open(UNIT=7,FILE='Results_blade_bending_moments.txt',ACCESS='SEQUENTIAL',STATUS=
    'REPLACE')
        do i=1,n_time_steps
            do j=1,n_horseshoe
                write(7,*) MB_ip(i,j), MB_op(i,j)
            end do
        end do
    close(7)

!created 13/04/2013
!validated
end subroutine write_outputs
!---------------------------------------------------------------
!subroutine for postprocessing the calculation and obtain the useful variables
```

```
subroutine postprocessing

    use horseshoe_locations
    use incident_speeds
    use aerodynamic_performance
    use operating_parameters
    use geometric_inputs
    use simulation_data
    use airfoil_polars
    use clock
    use incident_speeds


    implicit none

    integer i,j,b
    real(kind=8),parameter:: pi=4*atan(1.)
    real(kind=8):: aux_speed, vec_x(3),vec_y(3),vec_z(3), vec_L(n_time_steps,3,
    n_horseshoe),vec_D(n_time_steps,3,n_horseshoe)
    real (kind=8):: aux3(3), aux_phi(n_horseshoe),aux4(3)
    real aux_wind

    vec_x(:)=0.
    vec_y(:)=0.
    vec_z(:)=0.
    vec_x(1)=1.
    vec_y(2)=1.
    vec_z(3)=1.
    Q(:)=0
    T(:)=0
    P(:)=0
    MB_ip(:,:)=0.
    MB_op(:,:)=0.
    do i=1,n_time_steps
        do j=1,n_horseshoe
            do b=1,n_blades
                call spline(real(AOA(1,:)),real(Cd_pol(j,:)),alpha(blade_phase(b,i+1),j),
    Cd(blade_phase(b,i+1),j),n_data_points,1)
                call spline(real(AOA(1,:)),real(Cm_pol(j,:)),alpha(blade_phase(b,i+1),j),
    Cm(blade_phase(b,i+1),j),n_data_points,1)
            end do

            aux_speed=dot_product(el_vect1(i,:,j),total_wind(i,1:3,j))**2+dot_product
    (el_vect3(i,:,j),total_wind(i,1:3,j))**2
            L(i,j)=0.5*Cl(i,j)*density*area_distr(j)*aux_speed
            D(i,j)=0.5*Cd(i,j)*density*area_distr(j)*aux_speed
            Mz(i,j)=0.5*Cm(i,j)*density*area_distr(j)*aux_speed

            call crossprod(total_wind(i,1:3,j),el_vect2(i,:,j),vec_L(i,:,j))
            vec_L(i,:,j)=vec_L(i,:,j)/sqrt(dot_product(vec_L(i,:,j),vec_L(i,:,j)))
            !aux3(:,j)=vec_L(i,:,j)
            vec_D(i,:,j)=total_wind(i,1:3,j)/sqrt(dot_product(total_wind(i,1:3,j),
    total_wind(i,1:3,j)))
            !aux4(:,j)=vec_D(i,:,j)
        end do !j=1,n_horseshoe
     end do !i=1,n_time_steps   !the loop must be split up, the L and D must be obtained
    for all blades before calculating the Q,P and T

     open(UNIT=7,FILE="Cp_saved_pos.txt",ACCESS='SEQUENTIAL',STATUS='REPLACE')
        do i=1,n_horseshoe
            write(7,*) cp_saved_pos(1,:,i)
        end do
     close(7)

     do i=1,n_time_steps
        do j=1,n_horseshoe
            do b=1,n_blades
                Q(i)=Q(i)+L(blade_phase(b,i+1),j)*(dot_product(vec_L(i,:,j),vec_z)*
    cp_saved_pos(i,2,j)-dot_product(vec_L(i,:,j),vec_y)*cp_saved_pos(i,3,j))
                Q(i)=Q(i)+D(blade_phase(b,i+1),j)*(dot_product(vec_D(i,:,j),vec_z)*
    cp_saved_pos(i,2,j)-dot_product(vec_D(i,:,j),vec_y)*cp_saved_pos(i,3,j))

                T(i)=T(i)+D(blade_phase(b,i+1),j)*dot_product(vec_D(i,:,j),vec_x)
                T(i)=T(i)+L(blade_phase(b,i+1),j)*dot_product(vec_L(i,:,j),vec_x)
```

```
            end do

            do b=j+1,n_horseshoe
                call crossprod(cp_saved_pos(i,:,b)-cp_saved_pos(i,:,j),vec_L(i,:,b),aux3)
                call crossprod(cp_saved_pos(i,:,b)-cp_saved_pos(i,:,j),vec_D(i,:,b),aux4)
                MB_ip(i,j)=MB_ip(i,j)+L(i,b)*dot_product(vec_x,aux3)+D(i,b)*dot_product
    (vec_x,aux4)
                MB_op(i,j)=MB_op(i,j)+L(i,b)*dot_product(el_vect1(i,:,j),aux3)+D(i,b)*
    dot_product(el_vect1(i,:,j),aux4)
            end do

        end do
        !aux_wind=sum(wind(i,1,:))/n_horseshoe
        P(i)=Q(i)*omega
        Cp(i)=P(i)/(0.5*density*pi*(control_points_loc(1,n_horseshoe))**2*
    (wind_speed_module)**3)
        Ct(i)=T(i)/(0.5*density*pi*control_points_loc(1,n_horseshoe)**2*wind_speed_module*
    *2)
    end do

!created 21/04/2013
!modified 21/05/2013
!validated
end subroutine postprocessing
!-------------------------------------------------------------------------------------------
    ---------------
!subroutine with alternative version for calculating the blade self induced speed
subroutine blade_direct_induced_speed

    use horseshoe_locations
    use clock
    use incident_speeds
    use vortex_intensities
    use geometric_inputs
    use simulation_data

    implicit none

    integer i,j,b
    real(kind=8):: speeds(3),aux3(3),blade_ind_det(n_horseshoe,3,n_horseshoe)

    open(UNIT=7,FILE='Blade_ind_detail.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE')
    blade_ind(time_step,:,:)=0.d0
    do i=1,n_horseshoe


        do b=1,n_blades
            aux3(:)=0.d0
            do j=1,n_horseshoe

                call VLtfnc(temp_cp_loc(1,1,i),temp_cp_loc(1,2,i),temp_cp_loc(1,3,i),
    temp_hs_loc(b,1,j),temp_hs_loc(b,2,j),temp_hs_loc(b,3,j),temp_hs_loc(b,1,j+1),
    temp_hs_loc(b,2,j+1),temp_hs_loc(b,3,j+1),pr_attached_hs_int(blade_phase(b,time_step+
    1),j),speeds(1),speeds(2),speeds(3))

                aux3(1)=aux3(1)+speeds(1)
                aux3(2)=aux3(2)+speeds(2)
                aux3(3)=aux3(3)+speeds(3)

                blade_ind_det(i,1,j)=speeds(1)
                blade_ind_det(i,2,j)=speeds(2)
                blade_ind_det(i,3,j)=speeds(3)

                call VLtfnc(temp_cp_loc(1,1,i),temp_cp_loc(1,2,i),temp_cp_loc(1,3,i),
    temp_hs_loc(b,1,j+1),temp_hs_loc(b,2,j+1),temp_hs_loc(b,3,j+1),temp_hs_te_loc(b,1,j+1)
    ,temp_hs_te_loc(b,2,j+1),temp_hs_te_loc(b,3,j+1),pr_attached_hs_int(blade_phase(b,
    time_step+1),j),speeds(1),speeds(2),speeds(3))

                aux3(1)=aux3(1)+speeds(1)
                aux3(2)=aux3(2)+speeds(2)
                aux3(3)=aux3(3)+speeds(3)

                blade_ind_det(i,1,j)=blade_ind_det(i,1,j)+speeds(1)
                blade_ind_det(i,2,j)=blade_ind_det(i,2,j)+speeds(2)
```

```
                blade_ind_det(i,3,j)=blade_ind_det(i,3,j)+speeds(3)

                call VLtfnc(temp_cp_loc(1,1,i),temp_cp_loc(1,2,i),temp_cp_loc(1,3,i),
    temp_hs_te_loc(b,1,j+1),temp_hs_te_loc(b,2,j+1),temp_hs_te_loc(b,3,j+1),temp_hs_te_loc
    (b,1,j),temp_hs_te_loc(b,2,j),temp_hs_te_loc(b,3,j),pr_attached_hs_int(blade_phase(b,
    time_step+1),j),speeds(1),speeds(2),speeds(3))

                aux3(1)=aux3(1)+speeds(1)
                aux3(2)=aux3(2)+speeds(2)
                aux3(3)=aux3(3)+speeds(3)

                blade_ind_det(i,1,j)=blade_ind_det(i,1,j)+speeds(1)
                blade_ind_det(i,2,j)=blade_ind_det(i,2,j)+speeds(2)
                blade_ind_det(i,3,j)=blade_ind_det(i,3,j)+speeds(3)

                call VLtfnc(temp_cp_loc(1,1,i),temp_cp_loc(1,2,i),temp_cp_loc(1,3,i),
    temp_hs_te_loc(b,1,j),temp_hs_te_loc(b,2,j),temp_hs_te_loc(b,3,j),temp_hs_loc(b,1,j),
    temp_hs_loc(b,2,j),temp_hs_loc(b,3,j),pr_attached_hs_int(blade_phase(b,time_step+1),j)
    ,speeds(1),speeds(2),speeds(3))

                aux3(1)=aux3(1)+speeds(1)
                aux3(2)=aux3(2)+speeds(2)
                aux3(3)=aux3(3)+speeds(3)

                blade_ind_det(i,1,j)=blade_ind_det(i,1,j)+speeds(1)
                blade_ind_det(i,2,j)=blade_ind_det(i,2,j)+speeds(2)
                blade_ind_det(i,3,j)=blade_ind_det(i,3,j)+speeds(3)


            end do
            do j=1,n_horseshoe
                write(7,*) blade_ind_det(i,1,j),blade_ind_det(i,2,j),blade_ind_det(i,3,j)
            end do

            blade_ind(time_step,1,i)=blade_ind(time_step,1,i)+aux3(1)
            blade_ind(time_step,2,i)=blade_ind(time_step,2,i)+aux3(2)
            blade_ind(time_step,3,i)=blade_ind(time_step,3,i)+aux3(3)
        end do


    end do

    close(7)

end subroutine blade_direct_induced_speed
!-------------------------------------------------------------
!subroutine to run for single point simulations
subroutine single_point_simulation

    use simulation_data

    use horseshoe_locations
    use operating_parameters
    use geometric_inputs
    use wake_sheet
    use clock
    use aerodynamic_performance
    use file_names

    implicit none

    real, parameter:: pi=4.*atan(1.)

    open(UNIT=7,FILE=inputfile,ACCESS='SEQUENTIAL')
    read(7,*)
    read(7,*)
    read(7,*)
    read(7,*) n_blades
    read(7,*) hub_to_br
    read(7,*) pitch
    pitch=pitch*pi/180
    read(7,*) omega
    read(7,*) wind_speed_module
    read(7,*) density
```

```
    read(7,*) n_horseshoe
    read(7,*) n_wake_lines
    read(7,*) b_wake_roll_up

    v0(1)=wind_speed_module
    v0(2:3)=0.
    n_time_steps=1
    upflow_angle=0.
    yaw_misalignment=0.
    hub_height=50.
    exp_a=0.
    b_stationary_sim=1
    azimut=0.
    preazimut=azimut
    wake_loc_init=1
    wake_int_init=1


    call initialize

    call read_input

    call phase_between_blades

    call place_elements

    call extrude_polars

    call from_cartesian_to_polars

    call calculations

    call check

    call postprocessing

    call write_outputs


!created 25/04/2013
!validated 01/05/2013
end subroutine single_point_simulation
!---------------------------------------------------------------------
!subroutine for simulating transient conditions
subroutine full_cycle_simulation

    use simulation_data

    use horseshoe_locations
    use wake_sheet
    use operating_parameters
    use aerodynamic_performance
    use clock
    use geometric_inputs
    use file_names

    implicit none

    real, parameter:: pi=4.*atan(1.)

    open(UNIT=7,FILE=inputfile,ACCESS='SEQUENTIAL')
    read(7,*)
    read(7,*)
    read(7,*)
    read(7,*) n_blades
    read(7,*) hub_to_br
    read(7,*) pitch
    pitch=pitch*pi/180
    read(7,*) hub_height
    read(7,*) exp_a
    read(7,*) tilt_angle
    read(7,*) upflow_angle
    read(7,*) yaw_misalignment
    read(7,*) omega
```

```
    read(7,*) wind_speed_module
    read(7,*) density
    read(7,*) n_horseshoe
    read(7,*) n_wake_lines
    read(7,*) n_time_steps
    read(7,*) b_wake_roll_up
    !-----
    azimut=0.
    preazimut=azimut
    b_stationary_sim=0
    wake_int_init=1
    wake_loc_init=1
    upflow_angle=upflow_angle+tilt_angle
    upflow_angle=upflow_angle*pi/180.
    yaw_misalignment=yaw_misalignment*pi/180
    v0(1)=wind_speed_module*cos(upflow_angle)*cos(yaw_misalignment)
    v0(2)=wind_speed_module*sin(yaw_misalignment)
    v0(3)=wind_speed_module*sin(upflow_angle)

    call initialize

    call read_input

    call phase_between_blades

    call place_elements

    call extrude_polars

    call from_cartesian_to_polars

    call calculations

    call check

    call postprocessing

    call write_outputs



!created 01/05/2013
!validated
end subroutine full_cycle_simulation
!---------------------------------------------------------------
!subroutine for obtaining the optimum lambda for the specified geometry
subroutine find_optimal_lambda

    use horseshoe_locations
    use incident_speeds
    use wake_sheet
    use operating_parameters
    use aerodynamic_performance
    use clock
    use geometric_inputs
    use simulation_data
    use lambda_variables
    use file_names

    implicit none

    real R, min_lambda, max_lambda,lambda_step
    integer n_lambda
    real(kind=8),parameter:: pi=4*atan(1.)

    open(UNIT=7,FILE=inputfile,ACCESS='SEQUENTIAL')
    read(7,*)
    read(7,*)
    read(7,*)
    read(7,*) n_blades
    read(7,*) hub_to_br
    read(7,*) pitch
    pitch=pitch*pi/180
    read(7,*) hub_height
```

```
    read(7,*) exp_a
    read(7,*) tilt_angle
    read(7,*) upflow_angle
    read(7,*) yaw_misalignment
    read(7,*) omega
    read(7,*) density
    read(7,*) n_horseshoe
    read(7,*) n_wake_lines
    read(7,*) n_time_steps
    read(7,*) b_wake_roll_up
    read(7,*) min_lambda
    read(7,*) max_lambda
    read(7,*) n_lambda
    !----------------
    azimut=0.
    preazimut=azimut
    b_stationary_sim=0
    if ((upflow_angle.EQ.0).AND.(yaw_misalignment.EQ.0).AND.(exp_a.EQ.0)) then
        b_stationary_sim=1
        n_time_steps=1
    end if
    print*,b_stationary_sim,"b_stationary_sim"
    wake_int_init=1
    wake_loc_init=1
    upflow_angle=upflow_angle*pi/180.
    yaw_misalignment=yaw_misalignment*pi/180
    !-------------------------
    call initialize
    call read_input
    call phase_between_blades
    call place_elements
    call extrude_polars
    call from_cartesian_to_polars
    allocate(lambda(n_lambda))
    allocate(cp_lambda(n_lambda))
    allocate(ct_lambda(n_lambda))
    R=hs_loc_pol(1,n_horseshoe+1)

    do lambda_step=1,n_lambda
        lambda(lambda_step)=min_lambda+(lambda_step-1)*(max_lambda-min_lambda)/(n_lambda-
    1)
        wind_speed_module=omega*R/lambda(lambda_step)
        v0(1)=wind_speed_module*cos(upflow_angle)*cos(yaw_misalignment)
        v0(2)=wind_speed_module*sin(yaw_misalignment)
        v0(3)=wind_speed_module*sin(upflow_angle)

        call calculations
        call postprocessing
        cp_lambda(lambda_step)=sum(Cp(:))/n_time_steps
        ct_lambda(lambda_step)=sum(Ct(:))/n_time_steps

    end do

    open(UNIT=7,FILE='Results_cp_and_ct_lambda_curve.txt',ACCESS='SEQUENTIAL',STATUS=
    'REPLACE')
        do lambda_step=1,n_lambda
            write(7,*) lambda(lambda_step),cp_lambda(lambda_step),ct_lambda(lambda_step)
        end do
    close(7)

    call write_outputs


end subroutine find_optimal_lambda
!---------------------------------------------------------------
!simulation mode for obtaining the power curve for a fixed lambda
subroutine power_curve

    use horseshoe_locations
    use incident_speeds
    use wake_sheet
    use operating_parameters
    use aerodynamic_performance
    use clock
```

```
    use geometric_inputs
    use simulation_data
    use power_curve_variables
    use file_names

    implicit none

    real R, omega_min, omega_max, P_max, lambda, inc_pc_step
    real(kind=8),parameter:: pi=4*atan(1.)
    integer i

    open(UNIT=7,FILE=inputfile,ACCESS='SEQUENTIAL')
    read(7,*)
    read(7,*)
    read(7,*)
    read(7,*) n_blades
    read(7,*) hub_to_br
    read(7,*) pitch
    pitch=pitch*pi/180
    read(7,*) hub_height
    read(7,*) exp_a
    read(7,*) tilt_angle
    read(7,*) upflow_angle
    read(7,*) yaw_misalignment
    read(7,*) density
    read(7,*) n_horseshoe
    read(7,*) n_wake_lines
    read(7,*) n_time_steps
    read(7,*) b_wake_roll_up
    read(7,*) lambda
    read(7,*) omega_min
    read(7,*) omega_max
    read(7,*) P_max
    read(7,*) n_pc_steps

    !-------------------------------------------------------------
    if ((exp_a.EQ.0).AND.(upflow_angle.EQ.0).AND.(yaw_misalignment.EQ.0)) then
        b_stationary_sim=1
        n_time_steps=1
    else
        upflow_angle=upflow_angle*pi/180
        yaw_misalignment=yaw_misalignment*pi/180
    end if
    wake_int_init=1
    wake_loc_init=1
    azimut=0
    preazimut=0

    call initialize
    call read_input
    call phase_between_blades
    call place_elements
    call extrude_polars
    call from_cartesian_to_polars

    inc_pc_step=(omega_max-omega_min)/n_pc_steps
    allocate(pc_P(n_pc_steps+1))
    allocate(pc_T(n_pc_steps+1))
    allocate(pc_omega(n_pc_steps+1))
    allocate(pc_wspeedm(n_pc_steps+1))
    R=hs_loc_pol(1,n_horseshoe+1)
    i=1
    pc_P(:)=0.
    do while ((pc_P(i).LT.P_max).AND.(i.LT.n_pc_steps+1))
        omega=omega_min+(i-1)*inc_pc_step
        wind_speed_module=omega*R/lambda

        v0(1)=wind_speed_module*cos(upflow_angle)*cos(yaw_misalignment)
        v0(2)=wind_speed_module*sin(yaw_misalignment)
        v0(3)=wind_speed_module*sin(upflow_angle)

        call calculations
        call postprocessing
```

```
        pc_P(i)=sum(P(:))/n_time_steps
        pc_T(i)=sum(T(:))/n_time_steps
        pc_omega(i)=omega
        pc_wspeedm(i)=wind_speed_module

        i=i+1
    end do

    open(UNIT=7,FILE='Results_Power_Curve_performance.txt',ACCESS='SEQUENTIAL',STATUS=
    'REPLACE')
        do i=1,n_pc_steps
            write(7,*) pc_P(i), pc_T(i)
        end do
    close(7)
    open(UNIT=7,FILE='Results_Power_Curve_speeds.txt',ACCESS='SEQUENTIAL',STATUS='REPLACE'
    )
        do i=1,n_pc_steps
            write(7,*) pc_omega(i), pc_wspeedm(i)
        end do
    close(7)

end subroutine power_curve
!---------------------------------------------------------------------
subroutine place_wake_sheet  !former full_detail

    use horseshoe_locations
    use wake_sheet
    use incident_speeds
    use vortex_intensities
    use aerodynamic_performance
    use operating_parameters
    use clock
    use simulation_data
    use geometric_inputs
    implicit none

    integer i,j,k,b,m, aux_wl,aux_ts
    real azimut_blade,azimut_wake,mat_az(3,3),mat_pitch(3,3),mat_c(3,3)
    real (kind=8):: aux_te_loc(3,n_horseshoe+1)
    real (kind=8):: aux_nodes_loc(3,(n_horseshoe+1)*n_wake_lines), aux_nodes_speeds(3,
    (n_horseshoe+1)*n_wake_lines)
    real (kind=8):: aux3(3), aux_pos(3), aux_pos_w1(3),aux_pos_w2(3),u,v,w
    real (kind=8):: aux_x0_pos(n_time_steps,3,n_horseshoe+1), aux_speeds(3,n_horseshoe+1)

    if (wake_loc_init.EQ.1) then
        !this routine is only done if any wake model hasn't been built before during the
    current simulation
        !pause
        do i=1,n_time_steps
            azimut_blade=preazimut+omega*inc_time*(i-1)
            do j=1,n_wake_lines
                azimut_wake=azimut_blade-omega*inc_time*(j-1)

                mat_az(1,1)=1.
                mat_az(2,1)=0.
                mat_az(3,1)=0.
                mat_az(1,2)=0.
                mat_az(2,2)=-sin(azimut_wake)
                mat_az(3,2)=cos(azimut_wake)
                mat_az(1,3)=0.
                mat_az(2,3)=-cos(azimut_wake)
                mat_az(3,3)=-sin(azimut_wake)


                mat_pitch(2,1)=0.
                mat_pitch(1,2)=0.
                mat_pitch(2,2)=1.
                mat_pitch(3,2)=0.
                mat_pitch(2,3)=0.

                mat_c(1,1)=1.
                mat_c(2,1)=0.
                mat_c(3,1)=0.
                mat_c(1,2)=0.
```

```
                mat_c(2,2)=1.
                mat_c(3,2)=0.
                mat_c(1,3)=0.
                mat_c(2,3)=0.
                mat_c(3,3)=-1.

                mat_pitch(1,1)=cos(pitch+twist(1))
                mat_pitch(3,1)=sin(pitch+twist(1))
                mat_pitch(1,3)=-sin(pitch+twist(1))
                mat_pitch(3,3)=cos(pitch+twist(1))

                aux_pos(1)=0.
                aux_pos(2)=horseshoe_te_loc(1,1)
                aux_pos(3)=horseshoe_te_loc(2,1)
                aux_pos=matmul(mat_c,aux_pos)
                aux_pos=matmul(mat_pitch,aux_pos)
                pr_wake_nodes_loc(i,:,1+(j-1)*(n_horseshoe+1))=matmul(mat_az,aux_pos)
                pr_wake_nodes_loc(i,1,1+(j-1)*(n_horseshoe+1))=pr_wake_nodes_loc(i,1,1+(j-1)*(n_horseshoe+1))+(j-1)*inc_time*v0(1)
                pr_wake_nodes_loc(i,2,1+(j-1)*(n_horseshoe+1))=pr_wake_nodes_loc(i,2,1+(j-1)*(n_horseshoe+1))+(j-1)*inc_time*v0(2)
                pr_wake_nodes_loc(i,3,1+(j-1)*(n_horseshoe+1))=pr_wake_nodes_loc(i,3,1+(j-1)*(n_horseshoe+1))+(j-1)*inc_time*v0(3)

                mat_pitch(1,1)=cos(pitch+twist(n_horseshoe))
                mat_pitch(3,1)=sin(pitch+twist(n_horseshoe))
                mat_pitch(1,3)=-sin(pitch+twist(n_horseshoe))
                mat_pitch(3,3)=cos(pitch+twist(n_horseshoe))

                aux_pos(1)=0.
                aux_pos(2)=horseshoe_te_loc(1,n_horseshoe+1)
                aux_pos(3)=horseshoe_te_loc(2,n_horseshoe+1)
                aux_pos=matmul(mat_c,aux_pos)
                aux_pos=matmul(mat_pitch,aux_pos)
                !print*,aux_pos , "pre"
                pr_wake_nodes_loc(i,:,j*(n_horseshoe+1))=matmul(mat_az,aux_pos)
                !print*, pr_wake_nodes_loc(i,:,j*(n_horseshoe+1)),"post"
                pr_wake_nodes_loc(i,1,j*(n_horseshoe+1))=pr_wake_nodes_loc(i,1,j*(n_horseshoe+1))+(j-1)*inc_time*v0(1)
                pr_wake_nodes_loc(i,2,j*(n_horseshoe+1))=pr_wake_nodes_loc(i,2,j*(n_horseshoe+1))+(j-1)*inc_time*v0(2)
                pr_wake_nodes_loc(i,3,j*(n_horseshoe+1))=pr_wake_nodes_loc(i,3,j*(n_horseshoe+1))+(j-1)*inc_time*v0(3)

                do k=2,n_horseshoe
                    mat_pitch(1,1)=cos(pitch+0.5*(twist(k-1)+twist(k)))
                    mat_pitch(3,1)=sin(pitch+0.5*(twist(k-1)+twist(k)))
                    mat_pitch(1,3)=-sin(pitch+0.5*(twist(k-1)+twist(k)))
                    mat_pitch(3,3)=cos(pitch+0.5*(twist(k-1)+twist(k)))

                    aux_pos(1)=0.
                    aux_pos(2)=horseshoe_te_loc(1,k)
                    aux_pos(3)=horseshoe_te_loc(2,k)
                    aux_pos=matmul(mat_c,aux_pos)
                    aux_pos=matmul(mat_pitch,aux_pos)
                    pr_wake_nodes_loc(i,:,k+(j-1)*(n_horseshoe+1))=matmul(mat_az,aux_pos)
                    pr_wake_nodes_loc(i,1,k+(j-1)*(n_horseshoe+1))=pr_wake_nodes_loc(i,1,k+(j-1)*(n_horseshoe+1))+(j-1)*inc_time*v0(1)
                    pr_wake_nodes_loc(i,2,k+(j-1)*(n_horseshoe+1))=pr_wake_nodes_loc(i,2,k+(j-1)*(n_horseshoe+1))+(j-1)*inc_time*v0(2)
                    pr_wake_nodes_loc(i,3,k+(j-1)*(n_horseshoe+1))=pr_wake_nodes_loc(i,3,k+(j-1)*(n_horseshoe+1))+(j-1)*inc_time*v0(3)
                end do



            end do
        end do
    !connectivity of the nodes in the wake for defining the rings
    do i=1,n_horseshoe*(n_wake_lines-1)
        wake_nodes_connec(1,i)=i
        wake_nodes_connec(2,i)=i+int((i-0.5)/n_horseshoe)
        wake_nodes_connec(3,i)=i+int((i-0.5)/n_horseshoe)+1
        wake_nodes_connec(4,i)=i+int((i-0.5)/n_horseshoe)+1+n_horseshoe
```

```
            wake_nodes_connec(5,i)=i+int((i-0.5)/n_horseshoe)+2+n_horseshoe
        end do

        wake_loc_init=b_wake_roll_up+wake_loc_init
    else if (wake_loc_init.EQ.2) then
        !this calculation will be done only when the simulation time increases, since the position of the wake is defined by its previous positions and the vorticity of the blade and th
        !and the wake itself at the previous instant


        do i=1,n_wake_lines
            !free stream contribution
            call wind_speed_for_wake(pr_wake_nodes_loc(blade_phase(1,time_step+b_stationary_sim),:,(i-1)*(n_horseshoe+1)+1:i*(n_horseshoe+1)),aux_speeds)
            do j=1,n_horseshoe+1
                aux_nodes_speeds(1,(i-1)*(n_horseshoe+1)+j)=aux_speeds(1,j)
                aux_nodes_speeds(2,(i-1)*(n_horseshoe+1)+j)=aux_speeds(2,j)
                aux_nodes_speeds(3,(i-1)*(n_horseshoe+1)+j)=aux_speeds(3,j)
            end do

            !blade induced speed
            do b=1,n_blades
                do j=1,n_horseshoe+1
                    aux3(:)=0
                    aux_pos(1)=pr_wake_nodes_loc(blade_phase(b,time_step+b_stationary_sim),1,(i-1)*(n_horseshoe+1)+j)
                    aux_pos(2)=pr_wake_nodes_loc(blade_phase(b,time_step+b_stationary_sim),2,(i-1)*(n_horseshoe+1)+j)
                    aux_pos(3)=pr_wake_nodes_loc(blade_phase(b,time_step+b_stationary_sim),3,(i-1)*(n_horseshoe+1)+j)
                    do k=1,n_horseshoe
                        call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),temp_hs_te_loc(b,1,k),temp_hs_te_loc(b,2,k),temp_hs_te_loc(b,3,k),temp_hs_loc(b,1,k),temp_hs_loc(b,2,k),temp_hs_loc(b,3,k),pr_attached_hs_int(blade_phase(b,time_step+b_stationary_sim),k),u,v,w)
                        aux3(1)=aux3(1)+u
                        aux3(2)=aux3(2)+v
                        aux3(3)=aux3(3)+w
                        call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),temp_hs_loc(b,1,k),temp_hs_loc(b,2,k),temp_hs_loc(b,3,k),temp_hs_loc(b,1,k+1),temp_hs_loc(b,2,k+1),temp_hs_loc(b,3,k+1),pr_attached_hs_int(blade_phase(b,time_step+b_stationary_sim),k),u,v,w)
                        aux3(1)=aux3(1)+u
                        aux3(2)=aux3(2)+v
                        aux3(3)=aux3(3)+w
                        call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),temp_hs_loc(b,1,k+1),temp_hs_loc(b,2,k+1),temp_hs_loc(b,3,k+1),temp_hs_te_loc(b,1,k+1),temp_hs_te_loc(b,2,k+1),temp_hs_te_loc(b,3,k+1),pr_attached_hs_int(blade_phase(b,time_step+b_stationary_sim),k),u,v,w)
                        aux3(1)=aux3(1)+u
                        aux3(2)=aux3(2)+v
                        aux3(3)=aux3(3)+w
!                       call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),temp_hs_te_loc(b,1,k+1),temp_hs_te_loc(b,2,k+1),temp_hs_te_loc(b,3,k+1),temp_hs_te_loc(b,1,k),temp_hs_te_loc(b,2,k),temp_hs_te_loc(b,3,k),pr_attached_hs_int(blade_phase(b,time_step+b_stationary_sim),k),u,v,w)
!                       aux3(1)=aux3(1)+u
!                       aux3(2)=aux3(2)+v
!                       aux3(3)=aux3(3)+w
                    end do ! loop for k--> from 1 to n_horseshoe contribution of each attached ring
                    aux_nodes_speeds(1,(i-1)*(n_horseshoe+1)+j)=aux_nodes_speeds(1,(i-1)*(n_horseshoe+1)+j)+aux3(1)
                    aux_nodes_speeds(2,(i-1)*(n_horseshoe+1)+j)=aux_nodes_speeds(2,(i-1)*(n_horseshoe+1)+j)+aux3(2)
                    aux_nodes_speeds(3,(i-1)*(n_horseshoe+1)+j)=aux_nodes_speeds(3,(i-1)*(n_horseshoe+1)+j)+aux3(3)
                end do ! loop for j--> from 1 to n_horseshoe+1 effect on each node along the i spanwise wake line
            end do !loop for b=1,n_blades--> for adding the effect of all the blades

            !wake self induced speed
            do b=1,n_blades
```

```
            do j=1,n_horseshoe+1
                aux3(:)=0
                aux_pos(1)=pr_wake_nodes_loc(blade_phase(1,time_step+b_stationary_sim)
,1,(i-1)*(n_horseshoe+1)+j)
                aux_pos(2)=pr_wake_nodes_loc(blade_phase(1,time_step+b_stationary_sim)
,2,(i-1)*(n_horseshoe+1)+j)
                aux_pos(3)=pr_wake_nodes_loc(blade_phase(1,time_step+b_stationary_sim)
,3,(i-1)*(n_horseshoe+1)+j)
                do k=1,n_horseshoe*(n_wake_lines-1)
                    aux_pos_w1(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),1,wake_nodes_connec(2,k))
                    aux_pos_w1(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),2,wake_nodes_connec(2,k))
                    aux_pos_w1(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),3,wake_nodes_connec(2,k))
                    aux_pos_w2(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),1,wake_nodes_connec(3,k))
                    aux_pos_w2(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),2,wake_nodes_connec(3,k))
                    aux_pos_w2(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),3,wake_nodes_connec(3,k))
                    call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),aux_pos_w1(1),
aux_pos_w1(2),aux_pos_w1(3),aux_pos_w2(1),aux_pos_w2(2),aux_pos_w2(3),pr_wake_int
(blade_phase(b,time_step+b_stationary_sim),k),u,v,w)
                    aux3(1)=aux3(1)+u
                    aux3(2)=aux3(2)+v
                    aux3(3)=aux3(3)+w

                    aux_pos_w1(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),1,wake_nodes_connec(3,k))
                    aux_pos_w1(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),2,wake_nodes_connec(3,k))
                    aux_pos_w1(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),3,wake_nodes_connec(3,k))
                    aux_pos_w2(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),1,wake_nodes_connec(5,k))
                    aux_pos_w2(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),2,wake_nodes_connec(5,k))
                    aux_pos_w2(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),3,wake_nodes_connec(5,k))
                    call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),aux_pos_w1(1),
aux_pos_w1(2),aux_pos_w1(3),aux_pos_w2(1),aux_pos_w2(2),aux_pos_w2(3),pr_wake_int
(blade_phase(b,time_step+b_stationary_sim),k),u,v,w)
                    aux3(1)=aux3(1)+u
                    aux3(2)=aux3(2)+v
                    aux3(3)=aux3(3)+w

                    aux_pos_w1(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),1,wake_nodes_connec(5,k))
                    aux_pos_w1(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),2,wake_nodes_connec(5,k))
                    aux_pos_w1(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),3,wake_nodes_connec(5,k))
                    aux_pos_w2(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),1,wake_nodes_connec(4,k))
                    aux_pos_w2(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),2,wake_nodes_connec(4,k))
                    aux_pos_w2(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),3,wake_nodes_connec(4,k))
                    call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),aux_pos_w1(1),
aux_pos_w1(2),aux_pos_w1(3),aux_pos_w2(1),aux_pos_w2(2),aux_pos_w2(3),pr_wake_int
(blade_phase(b,time_step+b_stationary_sim),k),u,v,w)
                    aux3(1)=aux3(1)+u
                    aux3(2)=aux3(2)+v
                    aux3(3)=aux3(3)+w

                    aux_pos_w1(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),1,wake_nodes_connec(4,k))
                    aux_pos_w1(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),2,wake_nodes_connec(4,k))
                    aux_pos_w1(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),3,wake_nodes_connec(4,k))
                    aux_pos_w2(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
b_stationary_sim),1,wake_nodes_connec(2,k))
```

```
                        aux_pos_w2(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),2,wake_nodes_connec(2,k))
                        aux_pos_w2(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),3,wake_nodes_connec(2,k))
                        call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),aux_pos_w1(1),
    aux_pos_w1(2),aux_pos_w1(3),aux_pos_w2(1),aux_pos_w2(2),aux_pos_w2(3),pr_wake_int
    (blade_phase(b,time_step+b_stationary_sim),k),u,v,w)
                        aux3(1)=aux3(1)+u
                        aux3(2)=aux3(2)+v
                        aux3(3)=aux3(3)+w

                    end do !loop for k=1,n_horseshoe*(n_wake_lines-1)--> for adding the
    effect of all the wake rings
                    aux_nodes_speeds(1,(i-1)*(n_horseshoe+1)+j)=aux_nodes_speeds(1,(i-1)*
    (n_horseshoe+1)+j)+aux3(1)
                    aux_nodes_speeds(2,(i-1)*(n_horseshoe+1)+j)=aux_nodes_speeds(2,(i-1)*
    (n_horseshoe+1)+j)+aux3(2)
                    aux_nodes_speeds(3,(i-1)*(n_horseshoe+1)+j)=aux_nodes_speeds(3,(i-1)*
    (n_horseshoe+1)+j)+aux3(3)
                end do !loop for j=1,n_horseshoe+1--> for calculating the velocities on
    each wake node along a spanwise wake line
            end do !loop fo b=1,n_blades--> for adding the effect of the wakes of each
    blade
        end do !loop for i --> from 1 to n_wake_lines for calculating the effect on each
    wake line



        do i=(n_horseshoe+1)*n_wake_lines,n_horseshoe+2,-1
            pr_wake_nodes_loc(time_step,1,i)=pr_wake_nodes_loc(blade_phase(1,time_step+
    b_stationary_sim),1,i-(n_horseshoe+1))+inc_time*aux_nodes_speeds(1,i)
            pr_wake_nodes_loc(time_step,2,i)=pr_wake_nodes_loc(blade_phase(1,time_step+
    b_stationary_sim),2,i-(n_horseshoe+1))+inc_time*aux_nodes_speeds(2,i)
            pr_wake_nodes_loc(time_step,3,i)=pr_wake_nodes_loc(blade_phase(1,time_step+
    b_stationary_sim),3,i-(n_horseshoe+1))+inc_time*aux_nodes_speeds(3,i)
        end do !loop for i=identification number of the first node of the second line to
    the last of the last line, in order to update the position

        do i=1,n_horseshoe+1
            pr_wake_nodes_loc(time_step,1,i)=temp_hs_te_loc(1,1,i)
            pr_wake_nodes_loc(time_step,2,i)=temp_hs_te_loc(1,2,i)
            pr_wake_nodes_loc(time_step,3,i)=temp_hs_te_loc(1,3,i)
        end do

    end if

!created 19/01/2013
!not validated
end subroutine place_wake_sheet !former full_detail
!-----------------------------------------------------------------
!this subroutine is an attemp to reduce the simulation time needed to build the wake
    geometry
subroutine place_wake_sheet_reduced

    use horseshoe_locations
    use wake_sheet
    use incident_speeds
    use vortex_intensities
    use aerodynamic_performance
    use operating_parameters
    use clock
    use simulation_data
    use geometric_inputs
    implicit none

    integer i,j,k,b,m,n, aux_wl,aux_ts, lim1,lim2
    real azimut_blade,azimut_wake,mat_az(3,3),mat_pitch(3,3),mat_c(3,3)
    real (kind=8):: aux_te_loc(3,n_horseshoe+1)
    real (kind=8):: aux_nodes_loc(3,(n_horseshoe+1)*n_wake_lines), aux_nodes_speeds(3,
    (n_horseshoe+1)*n_wake_lines)
    real (kind=8):: aux3(3), aux_pos(3), aux_pos_w1(3),aux_pos_w2(3),u,v,w
    real (kind=8):: aux_x0_pos(n_time_steps,3,n_horseshoe+1), aux_speeds(3,n_horseshoe+1)

    if (wake_loc_init.EQ.1) then
```

```
    !this routine is only done if any wake model hasn't been built before during the current simulation
    !pause
    do i=1,n_time_steps
        azimut_blade=preazimut+omega*inc_time*(i-1)
        do j=1,n_wake_lines
            azimut_wake=azimut_blade-omega*inc_time*(j-1)

            mat_az(1,1)=1.
            mat_az(2,1)=0.
            mat_az(3,1)=0.
            mat_az(1,2)=0.
            mat_az(2,2)=-sin(azimut_wake)
            mat_az(3,2)=cos(azimut_wake)
            mat_az(1,3)=0.
            mat_az(2,3)=-cos(azimut_wake)
            mat_az(3,3)=-sin(azimut_wake)


            mat_pitch(2,1)=0.
            mat_pitch(1,2)=0.
            mat_pitch(2,2)=1.
            mat_pitch(3,2)=0.
            mat_pitch(2,3)=0.

            mat_c(1,1)=1.
            mat_c(2,1)=0.
            mat_c(3,1)=0.
            mat_c(1,2)=0.
            mat_c(2,2)=1.
            mat_c(3,2)=0.
            mat_c(1,3)=0.
            mat_c(2,3)=0.
            mat_c(3,3)=-1.

            mat_pitch(1,1)=cos(pitch+twist(1))
            mat_pitch(3,1)=sin(pitch+twist(1))
            mat_pitch(1,3)=-sin(pitch+twist(1))
            mat_pitch(3,3)=cos(pitch+twist(1))

            aux_pos(1)=0.
            aux_pos(2)=horseshoe_te_loc(1,1)
            aux_pos(3)=horseshoe_te_loc(2,1)
            aux_pos=matmul(mat_c,aux_pos)
            aux_pos=matmul(mat_pitch,aux_pos)
            pr_wake_nodes_loc(i,:,1+(j-1)*(n_horseshoe+1))=matmul(mat_az,aux_pos)
            pr_wake_nodes_loc(i,1,1+(j-1)*(n_horseshoe+1))=pr_wake_nodes_loc(i,1,1+(j-1)*(n_horseshoe+1))+(j-1)*inc_time*v0(1)
            pr_wake_nodes_loc(i,2,1+(j-1)*(n_horseshoe+1))=pr_wake_nodes_loc(i,2,1+(j-1)*(n_horseshoe+1))+(j-1)*inc_time*v0(2)
            pr_wake_nodes_loc(i,3,1+(j-1)*(n_horseshoe+1))=pr_wake_nodes_loc(i,3,1+(j-1)*(n_horseshoe+1))+(j-1)*inc_time*v0(3)

            mat_pitch(1,1)=cos(pitch+twist(n_horseshoe))
            mat_pitch(3,1)=sin(pitch+twist(n_horseshoe))
            mat_pitch(1,3)=-sin(pitch+twist(n_horseshoe))
            mat_pitch(3,3)=cos(pitch+twist(n_horseshoe))

            aux_pos(1)=0.
            aux_pos(2)=horseshoe_te_loc(1,n_horseshoe+1)
            aux_pos(3)=horseshoe_te_loc(2,n_horseshoe+1)
            aux_pos=matmul(mat_c,aux_pos)
            aux_pos=matmul(mat_pitch,aux_pos)
            !print*,aux_pos , "pre"
            pr_wake_nodes_loc(i,:,j*(n_horseshoe+1))=matmul(mat_az,aux_pos)
            !print*, pr_wake_nodes_loc(i,:,j*(n_horseshoe+1)),"post"
            pr_wake_nodes_loc(i,1,j*(n_horseshoe+1))=pr_wake_nodes_loc(i,1,j*(n_horseshoe+1))+(j-1)*inc_time*v0(1)
            pr_wake_nodes_loc(i,2,j*(n_horseshoe+1))=pr_wake_nodes_loc(i,2,j*(n_horseshoe+1))+(j-1)*inc_time*v0(2)
            pr_wake_nodes_loc(i,3,j*(n_horseshoe+1))=pr_wake_nodes_loc(i,3,j*(n_horseshoe+1))+(j-1)*inc_time*v0(3)

            do k=2,n_horseshoe
```

```
                    mat_pitch(1,1)=cos(pitch+0.5*(twist(k-1)+twist(k)))
                    mat_pitch(3,1)=sin(pitch+0.5*(twist(k-1)+twist(k)))
                    mat_pitch(1,3)=-sin(pitch+0.5*(twist(k-1)+twist(k)))
                    mat_pitch(3,3)=cos(pitch+0.5*(twist(k-1)+twist(k)))

                    aux_pos(1)=0.
                    aux_pos(2)=horseshoe_te_loc(1,k)
                    aux_pos(3)=horseshoe_te_loc(2,k)
                    aux_pos=matmul(mat_c,aux_pos)
                    aux_pos=matmul(mat_pitch,aux_pos)
                    pr_wake_nodes_loc(i,:,k+(j-1)*(n_horseshoe+1))=matmul(mat_az,aux_pos)
                    pr_wake_nodes_loc(i,1,k+(j-1)*(n_horseshoe+1))=pr_wake_nodes_loc(i,1,k
+(j-1)*(n_horseshoe+1))+(j-1)*inc_time*v0(1)
                    pr_wake_nodes_loc(i,2,k+(j-1)*(n_horseshoe+1))=pr_wake_nodes_loc(i,2,k
+(j-1)*(n_horseshoe+1))+(j-1)*inc_time*v0(2)
                    pr_wake_nodes_loc(i,3,k+(j-1)*(n_horseshoe+1))=pr_wake_nodes_loc(i,3,k
+(j-1)*(n_horseshoe+1))+(j-1)*inc_time*v0(3)
                end do



            end do
        end do
        !print*,pr_wake_nodes_loc(1,:,:)
       !pause
       !connectivity of the nodes in the wake for defining the rings
       do i=1,n_horseshoe*(n_wake_lines-1)
           wake_nodes_connec(1,i)=i
           wake_nodes_connec(2,i)=i+int((i-0.5)/n_horseshoe)
           wake_nodes_connec(3,i)=i+int((i-0.5)/n_horseshoe)+1
           wake_nodes_connec(4,i)=i+int((i-0.5)/n_horseshoe)+1+n_horseshoe
           wake_nodes_connec(5,i)=i+int((i-0.5)/n_horseshoe)+2+n_horseshoe
       end do

       wake_loc_init=b_wake_roll_up+wake_loc_init
   else if (wake_loc_init.EQ.2) then
       !pause
       !this calculation will be done only when the simulation time increases, since the
   position of the wake is defined by its previous positions and the vorticity of the
   blade and th
       !and the wake itself at the previous instant


       do i=1,n_wake_lines
           !free stream contribution
           call wind_speed_for_wake(pr_wake_nodes_loc(blade_phase(1,time_step+
   b_stationary_sim),:,(i-1)*(n_horseshoe+1)+1:i*(n_horseshoe+1)),aux_speeds)
           do j=1,n_horseshoe+1
               aux_nodes_speeds(1,(i-1)*(n_horseshoe+1)+j)=aux_speeds(1,j)
               aux_nodes_speeds(2,(i-1)*(n_horseshoe+1)+j)=aux_speeds(2,j)
               aux_nodes_speeds(3,(i-1)*(n_horseshoe+1)+j)=aux_speeds(3,j)
           end do

           !blade induced speed
           do b=1,n_blades
               do j=1,n_horseshoe+1
                   aux3(:)=0
                   aux_pos(1)=pr_wake_nodes_loc(blade_phase(b,time_step+b_stationary_sim)
   ,1,(i-1)*(n_horseshoe+1)+j)
                   aux_pos(2)=pr_wake_nodes_loc(blade_phase(b,time_step+b_stationary_sim)
   ,2,(i-1)*(n_horseshoe+1)+j)
                   aux_pos(3)=pr_wake_nodes_loc(blade_phase(b,time_step+b_stationary_sim)
   ,3,(i-1)*(n_horseshoe+1)+j)
                   do k=1,n_horseshoe
                       call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),temp_hs_te_loc(b,1,k)
   ,temp_hs_te_loc(b,2,k),temp_hs_te_loc(b,3,k),temp_hs_loc(b,1,k),temp_hs_loc(b,2,k),
   temp_hs_loc(b,3,k),pr_attached_hs_int(blade_phase(b,time_step+b_stationary_sim),k),u,v
   ,w)
                       aux3(1)=aux3(1)+u
                       aux3(2)=aux3(2)+v
                       aux3(3)=aux3(3)+w
                       call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),temp_hs_loc(b,1,k),
   temp_hs_loc(b,2,k),temp_hs_loc(b,3,k),temp_hs_loc(b,1,k+1),temp_hs_loc(b,2,k+1),
   temp_hs_loc(b,3,k+1),pr_attached_hs_int(blade_phase(b,time_step+b_stationary_sim),k),u
```

```
    ,v,w)
                        aux3(1)=aux3(1)+u
                        aux3(2)=aux3(2)+v
                        aux3(3)=aux3(3)+w
                        call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),temp_hs_loc(b,1,k+1),
    temp_hs_loc(b,2,k+1),temp_hs_loc(b,3,k+1),temp_hs_te_loc(b,1,k+1),temp_hs_te_loc(b,2,k
    +1),temp_hs_te_loc(b,3,k+1),pr_attached_hs_int(blade_phase(b,time_step+
    b_stationary_sim),k),u,v,w)
                        aux3(1)=aux3(1)+u
                        aux3(2)=aux3(2)+v
                        aux3(3)=aux3(3)+w
!                       call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),temp_hs_te_loc(b,1,k+
    1),temp_hs_te_loc(b,2,k+1),temp_hs_te_loc(b,3,k+1),temp_hs_te_loc(b,1,k),
    temp_hs_te_loc(b,2,k),temp_hs_te_loc(b,3,k),pr_attached_hs_int(blade_phase(b,time_step
    +b_stationary_sim),k),u,v,w)
!                       aux3(1)=aux3(1)+u
!                       aux3(2)=aux3(2)+v
!                       aux3(3)=aux3(3)+w
                    end do ! loop for k--> from 1 to n_horseshoe contribution of each
    attached ring
                    aux_nodes_speeds(1,(i-1)*(n_horseshoe+1)+j)=aux_nodes_speeds(1,(i-1)*
    (n_horseshoe+1)+j)+aux3(1)
                    aux_nodes_speeds(2,(i-1)*(n_horseshoe+1)+j)=aux_nodes_speeds(2,(i-1)*
    (n_horseshoe+1)+j)+aux3(2)
                    aux_nodes_speeds(3,(i-1)*(n_horseshoe+1)+j)=aux_nodes_speeds(3,(i-1)*
    (n_horseshoe+1)+j)+aux3(3)
                end do ! loop for j--> from 1 to n_horseshoe+1 effect on each node along
    the i spanwise wake line
            end do !loop for b=1,n_blades--> for adding the effect of all the blades

            !wake self induced speed
            !do b=1,n_blades
            b=1
                do j=1,n_horseshoe+1
                    aux3(:)=0
                    aux_pos(1)=pr_wake_nodes_loc(blade_phase(1,time_step+b_stationary_sim)
    ,1,(i-1)*(n_horseshoe+1)+j)
                    aux_pos(2)=pr_wake_nodes_loc(blade_phase(1,time_step+b_stationary_sim)
    ,2,(i-1)*(n_horseshoe+1)+j)
                    aux_pos(3)=pr_wake_nodes_loc(blade_phase(1,time_step+b_stationary_sim)
    ,3,(i-1)*(n_horseshoe+1)+j)
                    do n=-min(i-1,10),min(n_wake_lines-i,10)
                        lim1=max(i-1+n,0)*(n_horseshoe+1)+max(j-10,1)
                        lim2=max(i-1+n,0)*(n_horseshoe+1)+min(j+10,n_horseshoe+1)
                        do k=1,n_horseshoe*(n_wake_lines-1)
                            aux_pos_w1(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),1,wake_nodes_connec(2,k))
                            aux_pos_w1(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),2,wake_nodes_connec(2,k))
                            aux_pos_w1(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),3,wake_nodes_connec(2,k))
                            aux_pos_w2(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),1,wake_nodes_connec(3,k))
                            aux_pos_w2(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),2,wake_nodes_connec(3,k))
                            aux_pos_w2(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),3,wake_nodes_connec(3,k))
                            call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),aux_pos_w1(1),
    aux_pos_w1(2),aux_pos_w1(3),aux_pos_w2(1),aux_pos_w2(2),aux_pos_w2(3),pr_wake_int
    (blade_phase(b,time_step+b_stationary_sim),k),u,v,w)
                            aux3(1)=aux3(1)+u
                            aux3(2)=aux3(2)+v
                            aux3(3)=aux3(3)+w

                            aux_pos_w1(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),1,wake_nodes_connec(3,k))
                            aux_pos_w1(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),2,wake_nodes_connec(3,k))
                            aux_pos_w1(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),3,wake_nodes_connec(3,k))
                            aux_pos_w2(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),1,wake_nodes_connec(5,k))
                            aux_pos_w2(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),2,wake_nodes_connec(5,k))
```

```
                        aux_pos_w2(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),3,wake_nodes_connec(5,k))
                        call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),aux_pos_w1(1),
    aux_pos_w1(2),aux_pos_w1(3),aux_pos_w2(1),aux_pos_w2(2),aux_pos_w2(3),pr_wake_int
    (blade_phase(b,time_step+b_stationary_sim),k),u,v,w)
                        aux3(1)=aux3(1)+u
                        aux3(2)=aux3(2)+v
                        aux3(3)=aux3(3)+w

                        aux_pos_w1(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),1,wake_nodes_connec(5,k))
                        aux_pos_w1(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),2,wake_nodes_connec(5,k))
                        aux_pos_w1(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),3,wake_nodes_connec(5,k))
                        aux_pos_w2(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),1,wake_nodes_connec(4,k))
                        aux_pos_w2(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),2,wake_nodes_connec(4,k))
                        aux_pos_w2(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),3,wake_nodes_connec(4,k))
                        call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),aux_pos_w1(1),
    aux_pos_w1(2),aux_pos_w1(3),aux_pos_w2(1),aux_pos_w2(2),aux_pos_w2(3),pr_wake_int
    (blade_phase(b,time_step+b_stationary_sim),k),u,v,w)
                        aux3(1)=aux3(1)+u
                        aux3(2)=aux3(2)+v
                        aux3(3)=aux3(3)+w

                        aux_pos_w1(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),1,wake_nodes_connec(4,k))
                        aux_pos_w1(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),2,wake_nodes_connec(4,k))
                        aux_pos_w1(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),3,wake_nodes_connec(4,k))
                        aux_pos_w2(1)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),1,wake_nodes_connec(2,k))
                        aux_pos_w2(2)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),2,wake_nodes_connec(2,k))
                        aux_pos_w2(3)=pr_wake_nodes_loc(blade_phase(b,time_step+
    b_stationary_sim),3,wake_nodes_connec(2,k))
                        call VLtfnc(aux_pos(1),aux_pos(2),aux_pos(3),aux_pos_w1(1),
    aux_pos_w1(2),aux_pos_w1(3),aux_pos_w2(1),aux_pos_w2(2),aux_pos_w2(3),pr_wake_int
    (blade_phase(b,time_step+b_stationary_sim),k),u,v,w)
                        aux3(1)=aux3(1)+u
                        aux3(2)=aux3(2)+v
                        aux3(3)=aux3(3)+w

                    end do !loop for k=1,n_horseshoe*(n_wake_lines-1)--> for adding
    the effect of all the wake rings
                end do !loop for n ---> reducing the number of influencing nodes
                aux_nodes_speeds(1,(i-1)*(n_horseshoe+1)+j)=aux_nodes_speeds(1,(i-1)*
    (n_horseshoe+1)+j)+aux3(1)
                aux_nodes_speeds(2,(i-1)*(n_horseshoe+1)+j)=aux_nodes_speeds(2,(i-1)*
    (n_horseshoe+1)+j)+aux3(2)
                aux_nodes_speeds(3,(i-1)*(n_horseshoe+1)+j)=aux_nodes_speeds(3,(i-1)*
    (n_horseshoe+1)+j)+aux3(3)
            end do !loop for j=1,n_horseshoe+1--> for calculating the velocities on
    each wake node along a spanwise wake line
        !end do !loop fo b=1,n_blades--> for adding the effect of the wakes of each
    blade
    end do !loop for i --> from 1 to n_wake_lines for calculating the effect on each
    wake line



    do i=(n_horseshoe+1)*n_wake_lines,n_horseshoe+2,-1
        pr_wake_nodes_loc(time_step,1,i)=pr_wake_nodes_loc(blade_phase(1,time_step+
    b_stationary_sim),1,i-(n_horseshoe+1))+inc_time*aux_nodes_speeds(1,i)
        pr_wake_nodes_loc(time_step,2,i)=pr_wake_nodes_loc(blade_phase(1,time_step+
    b_stationary_sim),2,i-(n_horseshoe+1))+inc_time*aux_nodes_speeds(2,i)
        pr_wake_nodes_loc(time_step,3,i)=pr_wake_nodes_loc(blade_phase(1,time_step+
    b_stationary_sim),3,i-(n_horseshoe+1))+inc_time*aux_nodes_speeds(3,i)
    end do !loop for i=identification number of the first node of the second line to
    the last of the last line, in order to update the position
```

```
        do i=1,n_horseshoe+1
            pr_wake_nodes_loc(time_step,1,i)=temp_hs_te_loc(1,1,i)
            pr_wake_nodes_loc(time_step,2,i)=temp_hs_te_loc(1,2,i)
            pr_wake_nodes_loc(time_step,3,i)=temp_hs_te_loc(1,3,i)
        end do

    end if

!created 19/01/2013
!not validated
end subroutine place_wake_sheet_reduced
```