`porcelli_module.f90`: Porcelli Model for Triggering Sawtooth Crashes

Module written by Glenn Bateman and C.N. Nguyen
bateman@lehigh.edu
Lehigh University Physics Department, 16 Memorial Drive East, Bethlehem, PA 18015, USA

1 June 2006

This module is an implementation of the model by Franco Porcelli [1] to compute the conditions needed to trigger a sawtooth crash in tokamak plasmas.

```
!  Usage:
!  call porcelli_model (issawt, lprint, n_unit &
!       , mzones, rminor_q_1 &
!       , rminor_zone_center, beta_thermal_i, beta_thermal_e &
!       , beta_total, shear, elongation &
!       , rminor_zone_boundary, volume, area &
!       , grad_beta_thermal_i, grad_beta_fast &
!       , b_poloidal &
!       , ion_mass, b_toroidal, r_major, denel0, ti0 &
!       , efast, zfast &
!       , nerror )
!
!  Many of the input variables in the argument list of
!  subroutine porcelli_model are profile arrays
!  which are used to prepare the scalar variables needed to
!  call subroutine porcelli_conditions.
!
!..Output variables:
!  -----
!  issawt  = 1 indicates that a sawtooth crash is triggered (output)
!            0 indicates that a sawtooth crash is not triggered
!
!..Input variables:
!  -----
!  lprint  > 0 for diagnostic printout
!              larger values of lprint produce more diagnostic output
!  n_unit = output unit number for diagnostic printout
!
!  mzones  = number of zone centers in grid
!  rminor_q_1 = minor radius of q = 1 surface [m] (input)
!        NOTE: the radius of the q=1 surface must be computed externally
!
!        The following profiles are given on the zone center grid
!          j=1,mzones, where the magnetic axis is assumed to be
!            half way between the first two zone centers.
!
!  rminor_zone_center(j) = minor radius profile for zone center grid [m]
!
!  beta_thermal_i(j) = local ion thermal beta on zone center grid
!                    = n_i T_i / ( ( b_toroidal )^2 / ( 2 \mu_0 ) )
!                      where n_i is the ion density (all species)
!                            T_i is the ion temperature at zone center
!                            b_toroidal is defined below
!  beta_thermal_e(j) = local electron thermal beta on zone center grid
!                      where n_e is the electron density at zone center
```

```
!                            T_e is the electron temperature at zone center
!                            b_toroidal is defined below
!  beta_total(j)     = local total pressure on zone center grid
!                      Note, the total pressure includes fast ions
!  shear(j)   = magnetic shear profile on zone center grid
!    shear = r * ( d q / d r ) / q
!      where r = minor radius
!  elongation(j) = elongation of flux surfaces on zone center grid
!
!        Note that the following profiles are on the zone boundary grid
!          j=1,mzones+1, where the magnetic axis is assumed to be
!            on the j=2 zone boundary (rminor_zone_boundary(2) = 0.0)
!
!  rminor_zone_boundary(j) = minor radius profile for zone boundary grid [m]
!  volume(j) = volume of flux sufraces on the zone boundary grid [m^3]
!  area(j)   = cross sectional area of flux sufraces [m^2]
!               on the zone boundary grid
!  grad_beta_thermal_i(j) = d beta_thermal_i / d r  [1/m]
!                         on the zone boundary grid
!  grad_beta_fast(j) = d beta_fast / d r  [1/m] on the zone boundary grid
!                      beta_fast = ( fast ion and electron energy density )
!                                  / ( ( b_toroidal )^2 / ( 2 \mu_0 ) )
!  b_poloidal(j) = poloidal magnetic field on zone boundary grid [Tesla]
!                  at outboard edge of midplane
!
!    Scalar variables:
!  ion_mass   = average ion mass (amu)
!  b_toroidal = toroidal magnetic field at r_major [Tesla]
!  r_major    = major radius [m]
!  denel0     = electron density at magnetic axis 10^20 m^-3
!  ti0        = central ion temperature (keV)
!  efast      = energy of fast ions at q=1 surface (keV) (scalar)
!  zfast      = charge of fast ions at q=1 surface
!
!..Input/Output variable:
!  -----
!  nerror  = 0 on output if there are no errors
!          > 0 on output indicates that an error has occurred
!          = 1 could not allocate arrays used within porcelli_module
!
!----------------------------------------
!
! Note:  This file, porcelli_module.f90, can be converted into a LaTeX
! file, porcelli_module.tex, by using the python script s2tex.py
! that is included in the distribution files, by typing:
! python ./s2tex.py porcelli_module.f90
!
```

A complete description of the Porcelli model is given in reference [1], and the notation used in that paper will be used here. It is assumed, in the Porcelli model, that each sawtooth crash is triggered by the onset of an $m = 1$ mode. The macroscopic drive for this mode is associated with an effective potential energy functional, δW, given by

$$\delta W = \delta W_{\rm MHD} + \delta W_{\rm KO} + \delta W_{\rm fast}.$$

where $\delta W_{\rm MHD}$, the ideal MHD term, includes effects from toroidicity and plasma shaping and where $\delta W_{\rm KO}$

Table 1: Definition of Variables

parameter	units	definition
R	m	major radius
a	m	minor radius
B_{p}	T	poloidal magnetic field at the midplane
B_{T}	T	vacuum magnetic field at major radius R
κ_1		elongation of the $q = 1$ magnetic surface
r_1	m	minor radius (half-width) of the $q = 1$ magnetic surface
$\overline{r}_1$	m	average minor radius at $q = 1$ surface, $\overline{r}_1 \equiv \sqrt{\kappa_1}\, r_1$
ϵ_1		$\epsilon_1 \equiv \bar{r}_1/R$
l_{i1}		internal inductance at $q = 1$ surface, $l_{\mathrm{i1}} = \int_0^{r_1} B_p^2 da/B_p^2 A_1$, ($A_1$ = plasma cross-section of $q = 1$ surface)
p_{fast}	Nm^{-2}	fast ion pressure (total pressure - thermal pressure)
p_{i}	Nm^{-2}	ion thermal plasma pressure
$p(r_1)$	Nm^{-2}	total plasma pressure (thermal + fast ion) at the $q = 1$ surface
$< p >_1$	Nm^{-2}	volume-averaged total plasma pressure within the $q = 1$ surface
s_1		magnetic shear, $s_1 = r(dq/dr)/q$, at the $q = 1$ surface
γ_ρ	s^{-1}	$m = 1$ semi-collisional growth rate, $\gamma_\rho = 1.1 s_1^{6/7} \hat{\rho}^{4/7}/(\tau_{\mathrm{A}} S^{1/7})$
S		τ_R/τ_A at the $q = 1$ surface, where $\tau_R = 43.7\, T_{\mathrm{i0}}^{3/2} \varepsilon_1 r_1^2$
τ_{A}	s	Alfvén time, $\tau_{\mathrm{A}} = R\sqrt{3}/v_{\mathrm{A}}$, where $v_{\mathrm{A}} = \sqrt{B_{\mathrm{T}}^2/(\mu_0 \rho)}$ (ρ = ion mass density)
μ_0	Tm/A	permeability of free space, $4\pi \times 10^{-7}$ Tm/A
$\hat{\rho}$		normalized ion Larmor radius, $\hat{\rho} = 3.25 \times 10^{-3} \sqrt{T_{\mathrm{i0}} m_{\mathrm{i}}} \Big/ B_{\mathrm{T}} \varepsilon_1 r_1$, ($m_i$ = ion mass in amu)
ω_{*i}	s^{-1}	diamagnetic frequency, $\omega_{*i} = [1000 T_{\mathrm{i0}}(\mathrm{keV})/\mathrm{B_T} \varepsilon_1 \mathrm{r_1 p_i}]\,(dp_{\mathrm{i}}/dr)$ (T_{i0} = central ion temperature in keV)
ω_{Dh}	s^{-1}	fast ion precession frequency, $\omega_{\mathrm{Dh}} = 500 E_{\mathrm{fast}}(\mathrm{keV})/Z_{\mathrm{fast}} B_{\mathrm{T}} R \varepsilon_1 r_1$ (Z_{fast} = fast ion charge state)

and δW_{fast} are a consequence of kinetic effects arising from trapped thermal particles and fast ions, respectively. The effective potential is normalized as follows:

$$\delta\hat{W} \equiv -\frac{4\delta W}{s_1 \xi^2 \epsilon_1^2 R B^2} = \delta\hat{W}_{\mathrm{MHD}} + \delta\hat{W}_{\mathrm{KO}} + \delta\hat{W}_{\mathrm{fast}},$$

where R is the major radius, B is the magnetic field, s is the magnetic shear, ξ is the radial displacement of the magnetic axis, and ϵ is the inverse aspect ratio. The subscript 1 denotes that variables are evaluated at the radius of the $q = 1$ surface. Definitions of the variables and the notation used in this paper are listed in Table 1. A description of each contribution to $\delta\hat{W}$ is given below, followed by the list of the stability criteria in which these contributions are used.

1. **Ideal Term,** $\delta\hat{W}_{\mathrm{MHD}}$: The ideal $m = 1$ mode is unstable when δW_{MHD} is negative and kinetic effects are negligible. Porcelli's approximation for the ideal term is

$$\delta\hat{W}_{\mathrm{MHD}} = -\frac{9\pi}{s_1}\left(l_{\mathrm{i1}} - \frac{1}{2}\right)\epsilon_1^2(\beta_{\mathrm{p1}}^2 - \beta_{\mathrm{pc}}^2) - \frac{18\pi}{s_1}\left(l_{\mathrm{i1}} - \frac{1}{2}\right)^3\left(\frac{\kappa_1 - 1}{2}\right)^2 \tag{1}$$

   where $\beta_{\mathrm{p1}} = (2\mu_0/B_{\mathrm{p1}}^2)[\langle p\rangle_1 - p\,(r_1)]$, with $\langle p\rangle_1$ denoting the volume average pressure within the $q = 1$ surface; μ_0 is the permeability of free space; $\beta_{\mathrm{pc}} = 0.3(1 - 5r_1/3a)$, l_{i1} is the internal inductance (see Table 1); and $\sqrt{\kappa_1}$ is the elongation. The first term on the right hand side of Eq. 1 is the Bussac term [2] and the second term accounts for the shaping of non-circular plasma cross-section [3].

2. **Trapped Thermal Ions,** $\delta\hat{W}_{\mathrm{KO}}$: The Kruskal-Oberman term, which represents the stabilizing effects arising from trapped thermal ions [4], is given by

$$\delta\hat{W}_{\mathrm{KO}} = 0.6\beta_{\mathrm{i0}}\frac{\sqrt{\epsilon_1}}{s_1}\frac{5}{2}\int_0^1 x^{3/2}\frac{p_{\mathrm{i}}(x)}{p_{\mathrm{i0}}}dx$$

where $x = r/r_1$, $p_{\mathrm{i}}(x)$ is the ion pressure profile, p_{i0} is its peak value and β_{i0} is the peak ion toroidal beta.

3. **Fast Ion Stabilization:** δW_{fast}: The presence of fast ions in the plasma have been known to stabilize sawtooth oscillations (clearly demonstrated in 'on, off' auxiliary heating experiments [5]). The stabilization arising from fast ions is a consequence of the conservation of the third adiabatic invariant [6]. In the Porcelli model, the expression for the stabilizing effects of fast ions is

$$\delta\hat{W}_{\mathrm{fast}} = \frac{c_f\epsilon_1^{3/2}}{s_1}\left(\frac{-2\mu_0}{B_{\mathrm{p}}^2(r_1)}\int_0^1 x^{3/2}\frac{dp_{\mathrm{fast}}}{dx}dx\right)$$

where p_{fast} is the fast ion pressure and c_f depends on the ratio of the mode frequency to the fast particle precession frequency. For frequencies much less than the fast ion precession frequency, the value c_f reduces to unity.

The Porcelli model involves three criteria for the triggering of sawtooth crashes. (The equations below are numbered as they are in Ref. [1].) According to this model, a sawtooth crash is triggered when any one of the three conditions described below is satisfied:

1. Trapped fast ions are effective in stabilizing the internal kink mode when their precession drift-orbital period is much shorter than the magnetic perturbation time. The driving force for the internal kink instability overcomes the fast ion stabilization and the mode is triggered when the normalized plasma core potential energy perturbation, $\delta\hat{W}_{\mathrm{core}} = \delta\hat{W}_{\mathrm{MHD}} + \delta\hat{W}_{\mathrm{KO}}$, satisfies the inequality [Eq. (13) in Ref. [1]]

$$-\delta\hat{W}_{\mathrm{core}} > c_h\omega_{\mathrm{Dh}}\tau_A \qquad \text{(Eq. 13)}$$

where c_h is a constant, ω_{Dh} is the fast ion precession frequency, and $\tau_A = R\sqrt{3}/v_{\mathrm{A}}$ is the Alfvén time.

2. The growth of the internal kink mode can also be stabilized by diamagnetic rotation. Thus, a sawtooth crash will be triggered when the driving force for the internal kink mode overcomes the diamagnetic rotation stabilization [Eq. (14) in Ref. [1]]

$$-\delta\hat{W} > 0.5\omega_{*i}\tau_A \qquad \text{(Eq. 14)}$$

where ω_{*i} is the thermal ion diamagnetic drift frequency. When fast ions are not present, Eq. (14) is a subset of Eq. (13).

3. When the $m = 1$ growth rate is determined by effects within a narrow layer centered on the mode rational $q = 1$ magnetic surface, the mode is referred to as a resistive internal kink mode. The width of the narrow layer is determined by the largest of the relevant short scale lengths at the $q = 1$ radius — for example, the ion Larmor radius, the resistive scale length $[r/(sS)^{1/3}]$, or the electron inertial skin depth $[c/\omega_{pe}]$. Auxiliary heated tokamak plasmas are typically in the semi-collisional (or ion-kinetic) regime, in which the ion Larmor radius is larger than the resistive scale length, which in turn is larger than the electron inertial skin depth. In this regime, ion physics dominates. The condition for a sawtooth crash to be triggered by the resistive internal kink mode, for a plasma in the semi-collisional regime, is Eq. (15) in Ref. [1]]

$$0.5\omega_{*i}\tau_A > -\delta\hat{W} > -c_\rho\hat{\rho} \qquad \text{and} \qquad c_*\gamma_\rho > \omega_{*i} \qquad \text{(Eq. 15)}$$

where c_ρ and c_* are constants and γ_ρ is the growth rate of the resistive internal kink mode in the semi-collisional regime. Since this growth rate depends on the magnetic shear, the condition on the growth rate in Eq. (15) can be expressed in the form of a critical magnetic shear. When $|\delta\hat{W}| < \hat{\rho}$, where $\hat{\rho}$ is the ion Larmor radius normalized with respect to the radius of the $q = 1$ surface, the mode structure changes in nature from that of a global resistive internal kink mode to that of a drift-tearing mode localized near the $q = 1$ surface, which is normally stable because of kinetic layer effects at high plasma temperature [7]. Note that the conditions given by Eq. (14) and Eq. (15) cannot be satisfied simultaneously.

The values chosen for the constants, $c_h = 0.4$, $c_\rho = 1$, $c_* = 3$, and $c_f = 1$, are the same as those used in reference [1]. Eq. 15 has two parts, which, in this paper, are referred to as part (a) (the condition that brackets the range of $\delta\hat{W}$ in Eq. 15) and part (b) (the condition on the growth rate in Eq. 15). The quantity $c_*\gamma_\rho = 3\gamma_\rho$ is referred to as the effective growth rate of the ion-kinetic regime.

```
!---:----1----:----2----:----3----:----4----:----5----:----6----:----7-!
! Record of revisions
!
!  1 June 2006  Bateman    Implemented error checking to set nerror
!                          and improved documentation
!
! 20 Dec  2005  Bateman    Implemented changes suggested by
!                          F.M.M. Nave and F. Porcelli
!                          used beta_thermal_i / beta_i_max in cpf
!                          grad(beta_total) --> grad(beta_thermal_i)
!                          500.0 / zfast in zomegf
! 21 Sept 2005  Bateman    Complete revision of module from C. Nguyen
!
!---:----1----:----2----:----3----:----4----:----5----:----6----:----7-!
!
      MODULE porcelli_constants

      REAL, PARAMETER :: &
       shear_minimum = 0.10 & ! minimum allowed value of magnetic shear
     , ctrho  = 1.0 & ! multiplier for rho_star in Eq. (15)
     , ctstar = 3.0 & ! multiplier for gamma_rho in Eq. (15)
     , cwrat  = 1.0 & ! multiplier
     , chfast = 0.4 & ! multiplier c_h in Eq. (13)
     , zpi    = 3.141592653589793 &     ! pi
     , zmu0   = 1.256637061435917E-06 & ! magnetic permeability
     , zepslon = 1.0e-12
!
      INTEGER, PARAMETER :: &
       ifastswt  = 1 & !  on/off switch for testing equ. 13 (fast ion)
     , iflarmswt = 1 & !  on/off switch for testing equ. 14
                     ! (rotational effects)
     , ikinswt   = 1   !  on/off switch for testing equ. 15 (tearing nature)

!
!  shear_minimum = minimum allowed value for magnetic shear at q=1
!    (Note: Magnetic shear must be bounded away from zero
!           because shear_at_q1 is used to normalize delta W.
!           Hence, max ( shear_at_q1, shear_minimum ) is used
!           in the normalization of the components of delta W
!           but not in the computation of the growth rate of the
!           resistive m=1 mode.)
!  ctrho   = numerical multiplier for rho_star (equ 16 of reference)
!  ctstar  = numerical multiplier for c_star  (equ 16 of reference)
```

```
!  cwrat   = numerical multiplier depend on ratio between mode frequency
!            and fast particle precesion freq. (c_f in equ B.8 of ref)
!  chfast  = numerical multiplier for c_h (equ 16 of reference)
!  zpi     = pi
!  zmu0    = magnetic permeability
!  zepslon = small number
!
!  ifastswt  = on/off switch for testing equ. 13 (fast ion)
!  iflarmswt = on/off switch for testing equ. 14 (rotational effects)
!  ikinswt   = on/off switch for testing equ. 15 (tearing nature)

      END MODULE porcelli_constants
!
!---:----1----:----2----:----3----:----4----:----5----:----6----:----7-!
!
      MODULE porcelli_diagnostic_outer
!
!..module porcelli_diagnostic_outer contains scalar variables
!  that are computed in the outer routine of the Porcelli module.
!  These scalar variables are stored in a data module so that
!  they can be used in diagnostic output if needed.
!
       REAL :: &
         beta_i_max       & ! maximum thermal ion beta
       , beta_tot_max     & ! maximum total beta
       , pressure_at_q1   & ! total pressure at the q=1 surface
       , pressure_i_at_q1 & ! ion thermal pressure at the q=1 surface
       , volume_at_q1     & ! volume of the q=1 magnetic surface
       , pressure_avg_up_to_q1 & ! average total pressure up to q=1 surface
       , elongation_at_q1 & ! elongation q=1 surface
       , b_pol_at_q1      & ! poloidal field (Tesla) at q=1 surface
                             !   (midplane outboard)
       , shear_at_q1      & ! magnetic shear at q=1 surface
       , grad_norm_press_at_q1 & ! thermal pressure scale length at q=1
       , grad_norm_press_i_at_q1 & ! ion thermal pressure scale length at q=1
       , area_at_q1      & ! cross sectional area of flux surface at q=1
       , inductance_at_q1 & ! internal inductance at q=1
       , r_minor          & ! minor radius of the plasma
                            ! = rminor_zone_boundary(mzones+1)
       , cpf              & !
       , betapolf1          ! fast ion poloidal beta gradient parameter
!
      END MODULE porcelli_diagnostic_outer
!
!---:----1----:----2----:----3----:----4----:----5----:----6----:----7-!
!
      MODULE porcelli_diagnostic_inner
!
!..module porcelli_diagnostic_outer contains scalar variables
!  that are computed in the inner routine of the Porcelli module.
!  These scalar variables are stored in a data module so that
!  they can be used in diagnostic output if needed.
!
!  The following variables are computed and documented in
```

```
!  subroutine porcelli_conditions in the porcelli module
!
       real :: zcel,zbetapc,zbetap1,zcmhd,zr1avg,zdiffbeta2,zdenom
       real :: zdelwbu,zdelwel,zdelwmhd,zdelwko,zdelwfast,zdelw
       real :: ztalf,zomegf,zomegdi,zrhostar,zlundnum,zgamp,zdelwc
!
!..The following variables refer to terms in Eqs. 13, 14 and 15 in
!  F. Porcelli, D. Boucher and M.N. Rosenbluth
!  Plasma Phys. Control. Fusion 38 (1996) 2163-2186
!
REAL :: eq13L,  & ! left  term in Eq. 13
        eq13R,  & ! right term in Eq. 13
        eq14L,  & ! left  term in Eq. 14
        eq14R,  & ! right term in Eq. 14
        eq15aL, & ! left  term in Eq. 15a
        eq15aC, & ! central term in Eq. 15a
        eq15aR, & ! right term in Eq. 15a
        eq15bL, & ! left  term in Eq. 15b
        eq15bR    ! right term in Eq. 15b
!
! leq13 = .TRUE. only if a sawtooth crash is triggered by Eq. 13
!                indicating that fast ion effect are insufficient to
!                stabilize the internal kink mode
! leq14 = .TRUE. only if a sawtooth crash is triggered by Eq. 14
!                indicating that diamagnetic rotation of thermal ions
!                is insufficient to stabilize the internal kink mode
! leq15 = .TRUE. only if a sawtooth crash is triggered by Eq. 15
!                indicating that the internal resistive kink mode
!                grows faster than the thermal ion diamagnetic frequency
!
logical :: leq13 = .FALSE.  & ! a sawtooth crash is triggered by Eq. 13
         , leq14 = .FALSE.  & ! a sawtooth crash is triggered by Eq. 14
         , leq15 = .FALSE.    ! a sawtooth crash is triggered by Eq. 15
!
      END MODULE porcelli_diagnostic_inner
!
!---:----1----:----2----:----3----:----4----:----5----:----6----:----7-!
!
! The following Porcelli module for tests whether or not
!   a sawtooth crash should be triggered.
! If a sawtooth should be triggered, the module will return is_sawt = 1,
!   if not => is_sawt = 0
!
       MODULE porcelli_module
!
       IMPLICIT NONE
!
       CONTAINS
!
!*******
!
       SUBROUTINE porcelli_model (issawt, lprint, n_unit &
       , mzones, rminor_q_1 &
       , rminor_zone_center, beta_thermal_i, beta_thermal_e &
```

```
     , beta_total, shear, elongation &
     , rminor_zone_boundary, volume, area &
     , grad_beta_thermal_i, grad_beta_fast &
     , b_poloidal &
     , ion_mass, b_toroidal, r_major, denel0, ti0 &
     , efast, zfast &
     , nerror )
!
!  Many of the input variables in the argument list of
!  subroutine porcelli_model are profile arrays
!  which are used to prepare the scalar variables needed to
!  call subroutine porcelli_conditions.
!
!..Output variables:
!  -----
!  issawt  = 1 indicates that a sawtooth crash is triggered (output)
!            0 indicates that a sawtooth crash is not triggered
!
!..Input variables:
!  -----
!  lprint  > 0 for diagnostic printout
!              larger values of lprint produce more diagnostic output
!  n_unit = output unit number for diagnostic printout
!
!  mzones  = number of zone centers in grid
!  rminor_q_1 = minor radius of q = 1 surface [m] (input)
!        NOTE: the radius of the q=1 surface must be computed externally
!
!        The following profiles are given on the zone center grid
!          j=1,mzones, where the magnetic axis is assumed to be
!            half way between the first two zone centers.
!
!  rminor_zone_center(j) = minor radius profile for zone center grid [m]
!
!  beta_thermal_i(j) = local ion thermal beta on zone center grid
!                    = n_i T_i / ( ( b_toroidal )^2 / ( 2 \mu_0 ) )
!                      where n_i is the ion density (all species)
!                            T_i is the ion temperature at zone center
!                            b_toroidal is defined below
!  beta_thermal_e(j) = local electron thermal beta on zone center grid
!                      where n_e is the electron density at zone center
!                            T_e is the electron temperature at zone center
!                            b_toroidal is defined below
!  beta_total(j)     = local total pressure on zone center grid
!                      Note, the total pressure includes fast ions
!  shear(j)   = magnetic shear profile on zone center grid
!    shear = r * ( d q / d r ) / q
!      where r = minor radius
!  elongation(j) = elongation of flux surfaces on zone center grid
!
!        Note that the following profiles are on the zone boundary grid
!          j=1,mzones+1, where the magnetic axis is assumed to be
!            on the j=2 zone boundary (rminor_zone_boundary(2) = 0.0)
!
```

```
!  rminor_zone_boundary(j) = minor radius profile for zone boundary grid [m]
!  volume(j) = volume of flux sufraces on the zone boundary grid [m^3]
!  area(j)   = cross sectional area of flux sufraces [m^2]
!                on the zone boundary grid
!  grad_beta_thermal_i(j) = d beta_thermal_i / d r  [1/m]
!                         on the zone boundary grid
!  grad_beta_fast(j) = d beta_fast / d r  [1/m] on the zone boundary grid
!                     beta_fast = ( fast ion and electron energy density )
!                                 / ( ( b_toroidal )^2 / ( 2 \mu_0 ) )
!  b_poloidal(j) = poloidal magnetic field on zone boundary grid [Tesla]
!                 at outboard edge of midplane
!
!    Scalar variables:
!  ion_mass   = average ion mass (amu)
!  b_toroidal = toroidal magnetic field at r_major [Tesla]
!  r_major    = major radius [m]
!  denel0     = electron density at magnetic axis 10^20 m^-3
!  ti0        = central ion temperature (keV)
!  efast      = energy of fast ions at q=1 surface (keV) (scalar)
!  zfast      = charge of fast ions at q=1 surface
!
! Output variable
! --------------_
!
!  nerror     = 0 for a normal return with no errors found
!             = 1 indicates that module was unable to allocate an array
!             = 2 indicates that mzones < 4, not enough elements in arrays
!             = 3 indicates that rminor_q_1 < rminor_zone_center(2)
!                           or rminor_q_1 > rminor_zone_boundary(mzones)
!                  (the q=1 surface is not in the plasma)
!             = 4 indicates that r_major < r_minor
!
!-----------------------------------------
!
      USE porcelli_constants
!
      USE porcelli_diagnostic_outer
!
!..module porcelli_diagnostic_outer contains the following scalar variables
!  that are computed in the outer routine of the Porcelli module
!
!  beta_i_max              maximum thermal ion beta
!  beta_tot_max            maximum total beta
!  pressure_at_q1          total pressure at the q=1 surface
!  pressure_i_at_q1        ion thermal pressure at the q=1 surface
!  volume_at_q1            volume of the q=1 magnetic surface
!  pressure_avg_up_to_q1   average total pressure up to q=1 surface
!  elongation_at_q1        elongation q=1 surface
!  b_pol_at_q1             poloidal field (Tesla) at q=1 surface
!                          (midplane outboard)
!  shear_at_q1             magnetic shear at q=1 surface
!  grad_norm_press_at_q1   thermal pressure scale length at q=1
!  grad_norm_press_i_at_q1 ion thermal pressure scale length at q=1
!  area_at_q1              cross sectional area of flux surface at q=1
```

```
!  inductance_at_q1        internal inductance at q=1
!  r_minor                 minor radius of the plasma
!                          = rminor_zone_boundary(mzones+1)
!  cpf
!  betapolf1               fast ion poloidal beta gradient parameter
!
!
      IMPLICIT NONE
!
      INTEGER, INTENT(IN) :: lprint, n_unit
!
      INTEGER, INTENT(IN) :: mzones
!
      INTEGER, INTENT(OUT) :: issawt
!
      REAL, INTENT(IN) :: rminor_q_1
!
      REAL, dimension(:), INTENT(IN) :: &
        rminor_zone_center, beta_thermal_i, beta_thermal_e &
      , beta_total, shear, elongation
!
      REAL, dimension(:), INTENT(IN) :: &
        rminor_zone_boundary, volume, area &
      , grad_beta_thermal_i, grad_beta_fast, b_poloidal
!
      REAL, INTENT(IN) :: &
        ion_mass, b_toroidal, r_major, ti0, denel0, efast, zfast
!
      INTEGER, INTENT(INOUT) :: nerror
!
      INTEGER :: ierror    ! internal error condition
!
      INTEGER :: jz
!
!..temporary arrays
!
      REAL, dimension(3) :: ztemp_in, ztemp_out
!
      REAL, dimension(:), allocatable :: ztemp1, ztemp2
!
      REAL, dimension(:), allocatable :: ceqtmp
!
!..Initialize
!
      issawt = 0
      nerror = 0
!
!..allocate temporary arrays
!
      ierror = 0
      ALLOCATE ( ztemp1(1:mzones), ztemp2(1:mzones) &
        , ceqtmp(1:4*mzones), STAT = ierror  )
!
      IF ( ierror > 0 .AND. lprint > 0 ) THEN
```

```
          WRITE (n_unit,*) ' After allocate, ierror = ', ierror
        ENDIF
!
        IF ( ierror > 0 ) THEN
          nerror = 1
          RETURN
        ENDIF
!
!..diagnostic output
!
        if ( lprint > 5 ) then
!
          WRITE(n_unit,*)
          WRITE(n_unit,101)
 101      FORMAT('  jz',t7,'rminor_zone_cent'  &
        , t25, 'beta_thermal_i', t41,'beta_thermal_e'  &
        , t58, 'beta_total', t74, 'shear', t90, 'elongation')
!
          do jz=1,mzones
            WRITE(n_unit,102) jz, rminor_zone_center(jz)  &
            , beta_thermal_i(jz), beta_thermal_e(jz), beta_total(jz)  &
            , shear(jz), elongation(jz)
          enddo
!
 102      FORMAT(i4,1p6e16.5)
!
          WRITE(n_unit,*)
          WRITE(n_unit,103)
 103      FORMAT('  jz',t6,'rminor_zone_boundary'  &
        , t28, 'volume', t44,'area'  &
        , t55, 'grad_beta_thermal_i'  &
        , t74, 'grad_beta_fast'  &
        , t90, 'b_poloidal')
!
          do jz=1,mzones
            WRITE(n_unit,102) jz, rminor_zone_boundary(jz) &
            , volume(jz), area(jz) &
            , grad_beta_thermal_i(jz), grad_beta_fast(jz) &
            , b_poloidal(jz)
          enddo
!
          WRITE(n_unit,*)
          WRITE(n_unit,*) ' rminor_q_1=', rminor_q_1 &
            , ' r_major=', r_major
!
          WRITE(n_unit,*)
          WRITE(n_unit,*) &
              ' ion_mass=', ion_mass &
            , ' b_toroidal=', b_toroidal
!
          WRITE(n_unit,*) ' denel0=', denel0 &
            , ' ti0=', ti0 &
            , ' efast=', efast, '  zfast = ', zfast
!
```

```
      endif
!
!..Check input variables
!
      IF ( mzones < 4 ) THEN
        nerror = 2
        RETURN
      ENDIF
!
      IF ( rminor_q_1 < rminor_zone_center(2) .or. &
           rminor_q_1 > rminor_zone_boundary(mzones) ) THEN
        nerror = 3
        RETURN
      ENDIF
!
!..compute, beta_i_max, the maximum thermal ion beta
!
      beta_i_max = 0.0
      beta_tot_max=0.0
      do jz = 1,mzones
         beta_i_max = max (  beta_i_max, beta_thermal_i(jz) )
         beta_tot_max = max (  beta_tot_max, beta_total(jz) )
      enddo
!
!..Minor radius of plasma
!
      r_minor = rminor_zone_boundary(mzones)
!
      IF ( r_major < r_minor ) THEN
        nerror = 4
        RETURN
      ENDIF
!
!  Interpolate the total pressure to the q=1 surface
!
      ztemp_in(1) = 0.0
      ztemp_in(2) = rminor_q_1
      ztemp_in(3) = rminor_zone_center(mzones-3)
!
      ztemp_out   = 0.0
!
      CALL cubint (rminor_zone_center, beta_total &
            ,mzones-1, 0, ceqtmp, mzones &
            ,ztemp_in, ztemp_out, 2, 0, 0.0, 1 &
       ,'abort computing total pressure at q=1 in sbrtn sawtst')
!
        pressure_at_q1 = ztemp_out(2) * b_toroidal**2 / ( 2.0 * zmu0 )
          ! total pressure nt/m**2 at q=1 surface
!
!  Interpolate the ion pressure to the q=1 surface
!
      ztemp_in(1) = 0.0
      ztemp_in(2) = rminor_q_1
      ztemp_in(3) = rminor_zone_center(mzones-3)
```

```
!
       ztemp_out   = 0.0
!
       CALL cubint (rminor_zone_center, beta_thermal_i &
             ,mzones-1, 0, ceqtmp, mzones &
             ,ztemp_in, ztemp_out, 2, 0, 0.0, 1 &
        ,'abort computing total pressure at q=1 in sbrtn sawtst')
!
         pressure_i_at_q1 = ztemp_out(2) * b_toroidal**2 / ( 2.0 * zmu0 )
           ! total pressure nt/m**2 at q=1 surface
!
!..compute volume of the q=1 surface
!
       ztemp_in(1) = 0.0
       ztemp_in(2) = rminor_q_1
       ztemp_in(3) = rminor_zone_center(mzones-3)
!
       ztemp_out   = 0.0
!
       CALL cubint (rminor_zone_center, volume &
             ,mzones-1, 0, ceqtmp, mzones &
             ,ztemp_in, ztemp_out, 2, 0, 0.0, 1 &
             ,'abort computing volume at q=1 in sbrtn sawtst')
!
       volume_at_q1 = ztemp_out(2)
!
! compute pressure_avg_up_to_q1 = average total pressure up to q=1 surface
!    (Nt/m^3)
!
!..integrate over volume using sbrtn cubint
!
!
!..first, interpolate the volume array from zone boundaries to zone centers
!
       ztemp1 = 0.0
!
       CALL cubint ( rminor_zone_boundary, volume &
             , mzones-1, 0, ceqtmp, mzones &
             , rminor_zone_center, ztemp1, mzones-1, 0, 0.0, 1 &
        , 'abort computing volume on zone centers in porcelli_module')
!
       ztemp1(1) = - ztemp1(2)
!
       ztemp_in(1) = 0.0
       ztemp_in(2) = volume_at_q1
       ztemp_in(3) = volume(mzones-3)
!
       ztemp_out   = 0.0
!
       CALL cubint (ztemp1, beta_total &
             ,mzones-1, 0, ceqtmp, mzones &
             ,ztemp_in, ztemp_out, 2, 3, 0.0, 1 &
             ,'abort computing sum vol*press at q=1 in porcelli_module')
!
```

```
      pressure_avg_up_to_q1 = ztemp_out(2) * b_toroidal**2 &
       / ( 2.0 * zmu0 * volume_at_q1 )
!
! compute elongation at q=1 surface
!
      ztemp_in(1) = 0.0
      ztemp_in(2) = rminor_q_1
      ztemp_in(3) = rminor_zone_center(mzones-3)
!
      ztemp_out   = 0.0
!
      CALL cubint (rminor_zone_center, elongation &
            ,mzones-1, 0, ceqtmp, mzones &
            ,ztemp_in, ztemp_out, 2, 0, 0.0, 1 &
        ,'abort computing elongation at q=1 in sbrtn sawtst')
!
      elongation_at_q1 = ztemp_out(2) ! elongation of q=1 surface
!
! compute b_pol_at_q1 = poloidal field (Tesla) at q=1 surface
!   (midplane outboard)
!
      ztemp_in(1) = 0.0
      ztemp_in(2) = rminor_q_1
      ztemp_in(3) = rminor_zone_center(mzones-3)
!
      ztemp_out   = 0.0
!
      CALL cubint (rminor_zone_center, b_poloidal &
            ,mzones-1, 0, ceqtmp, mzones &
            ,ztemp_in, ztemp_out, 2, 0, 0.0, 1 &
         ,'abort computing B-poloidal at q=1 in sbrtn sawtst')
!
      b_pol_at_q1 = ztemp_out(2)
!
! compute shear_at_q1 = magnetic shear at q=1 surface
!
      ztemp_in(1) = 0.0
      ztemp_in(2) = rminor_q_1
      ztemp_in(3) = rminor_zone_center(mzones-3)
!
      ztemp_out   = 0.0
!
      CALL cubint (rminor_zone_center, shear &
            ,mzones-1, 0, ceqtmp, mzones &
            ,ztemp_in, ztemp_out, 2, 0, 0.0, 1 &
            ,'abort computing shear at q=1 in sbrtn sawtst')
!
      shear_at_q1 = abs(ztemp_out(2))
!
! compute 1. / pressure gradient scale length at the q=1 surface
!
      ztemp_in(1) = 0.0
      ztemp_in(2) = rminor_q_1
      ztemp_in(3) = rminor_zone_center(mzones-3)
```

```
!
      ztemp_out   = 0.0
!
      CALL cubint (rminor_zone_center, grad_beta_thermal_i &
            ,mzones-1, 0, ceqtmp, mzones &
            ,ztemp_in, ztemp_out, 2, 0, 0.0, 1 &
            ,'abort computing press. length at q=1 in sbrtn sawtst')
!
      grad_norm_press_i_at_q1 = abs(ztemp_out(2) / pressure_i_at_q1 ) &
         * b_toroidal**2 / ( 2.0 * zmu0 )
      if ( grad_norm_press_i_at_q1 > 1. / zepslon) &
         grad_norm_press_i_at_q1 = 1. / rminor_q_1          ! default
!
! compute inductance_at_q1 = internal inductance at q=1
!
      ztemp_in(1) = 0.0
      ztemp_in(2) = rminor_q_1
      ztemp_in(3) = rminor_zone_center(mzones-3)
!
      ztemp_out   = 0.0
!
      CALL cubint (rminor_zone_center, area &
            ,mzones-1, 0, ceqtmp, mzones &
            ,ztemp_in, ztemp_out, 2, 0, 0.0, 1 &
            ,'abort computing sum area at q=1 in sbrtn sawtst')
!
      area_at_q1 = ztemp_out(2)
!
      DO jz=1,mzones-1
        ztemp1(jz) = area(jz)
        ztemp2(jz) = b_poloidal(jz)**2
      ENDDO
      ztemp1(1) = - ztemp1(3)
!
      ztemp_in(1) = 0.0
      ztemp_in(2) = area_at_q1
      ztemp_in(3) = area(mzones-3)
!
      ztemp_out   = 0.0
!
      CALL cubint ( ztemp1, ztemp2 &
            ,mzones-1, 0, ceqtmp, mzones &
            ,ztemp_in, ztemp_out, 2, 3, 0.0, 1 &
            ,'abort computing sum bpol**2da at q=1 in porcelli_module')
!
      inductance_at_q1 = ztemp_out(2) &
        / ( area_at_q1 * b_pol_at_q1**2 )
!
```

compute

$$\mathtt{cpf} = \frac{(5/2)\int_0^1 x^{3/2} P_{\rm thermal}(x) dx}{P_{\rm i0}}$$

```
!
!..integrate over radius using sbrtn cubint
```

```
!
       ztemp1 = 0.0
       DO jz=1,mzones
!
         ztemp1(jz) = &
           (abs( rminor_zone_boundary(jz) / rminor_q_1 ))**1.5 &
           * ( ( beta_thermal_i(jz) ) &
           / ( beta_i_max * rminor_q_1 ) )
!
       ENDDO
!
       ztemp_in(1) = 0.0
       ztemp_in(2) = rminor_q_1
       ztemp_in(3) = rminor_zone_center(mzones-3)
!
       ztemp_out   = 0.0
!
       CALL cubint (rminor_zone_boundary, ztemp1 &
             ,mzones-1, 0, ceqtmp, mzones &
             ,ztemp_in, ztemp_out, 2, 3, 0.0, 1 &
             ,'abort computing sum integ press at q=1 in porcelli module')
!
       cpf = 2.5 * ztemp_out(2)
!
```

compute

$$\texttt{zbetapolf1} = \frac{-2\mu_0}{B_{\rm p}^2}\int_0^1 x^{3/2}\frac{dp_{\rm fast}}{dx}dx$$

```
!
         IF ( efast < zepslon ) then
!
           betapolf1 = 0.0
!
         ELSE
!
!..integrate over radius using sbrtn cubint
!
           ztemp1 = 0.0
           DO jz=1,mzones
!
              ztemp1(jz) = &
                (abs( rminor_zone_boundary(jz) / rminor_q_1 ))**1.5 &
                           * grad_beta_fast(jz)
!
           ENDDO
!
           ztemp_in(1) = 0.0
           ztemp_in(2) = rminor_q_1
           ztemp_in(3) = rminor_zone_center(mzones-3)
!
           ztemp_out   = 0.0
!
           CALL cubint (rminor_zone_boundary, ztemp1 &
                ,mzones-1, 0, ceqtmp, mzones &
```

```
                ,ztemp_in, ztemp_out, 2, 3, 0.0, 1 &
                ,'abort computing sum integ press at q=1 in porcelli module')
!
            betapolf1 = - ztemp_out(2) * b_toroidal**2 / b_pol_at_q1**2
!
          ENDIF
!
! diagnostic printout
!
    IF ( lprint > 1 ) THEN
      WRITE (6,*)
      WRITE (6,*) 'cpf=',cpf,' elongation_at_q1=',elongation_at_q1
      WRITE (n_unit,*) 'ion_mass=',ion_mass
      WRITE (6,*) 'inductance_at_q1=',inductance_at_q1
      WRITE (n_unit,*)' betapolf1=',betapolf1,' beta_i_max=',beta_i_max
      WRITE (6,*) 'shear_at_q1=',shear_at_q1
      WRITE (n_unit,*)' r_major=',r_major &
       ,'  r_minor=', r_minor
      WRITE (6,*) 'rminor_q_1=',rminor_q_1
      WRITE (n_unit,*)' grad_norm_press_i_at_q1=' &
       , grad_norm_press_i_at_q1
      WRITE (n_unit,*)' denel0=',denel0,' ti0=',ti0
      WRITE (6,*) 'b_toroidal=',b_toroidal &
       ,' b_pol_at_q1=',b_pol_at_q1
      WRITE (6,*) 'efast=',efast, '  zfast=', zfast
      WRITE (n_unit,*)' pressure_avg_up_to_q1=',pressure_avg_up_to_q1 &
       ,' pressure_at_q1=',pressure_at_q1
      WRITE (n_unit,*) ' beta_tot_max = ', beta_tot_max &
       , '  beta_i_max = ', beta_i_max
      WRITE (6,*) 'ifastswt=',ifastswt,' iflarmswt=',iflarmswt &
       ,'ikinswt=',ikinswt
      WRITE (6,*)
    ENDIF
!
!..deallocate temporary arrays
!
       ierror = 0
       DEALLOCATE ( ztemp1, ztemp2, ceqtmp, STAT=ierror )
!
       IF ( ierror > 0 ) THEN
         WRITE (n_unit,*) ' After deallocate, ierror = ', ierror
       ENDIF
!
! r_minor = minor radius (half-width) at edge of plasma
!
          CALL porcelli_conditions ( issawt, lprint, n_unit &
           , cpf &
           , elongation_at_q1, ion_mass, inductance_at_q1 &
           , betapolf1, beta_i_max, shear_at_q1 &
           , r_major, r_minor, rminor_q_1 &
           , grad_norm_press_i_at_q1, denel0 &
           , b_toroidal, b_pol_at_q1, ti0, efast, zfast &
           , pressure_avg_up_to_q1, pressure_at_q1 )
!
```

```
      RETURN
!
      END SUBROUTINE porcelli_model
!-----------------------------------------------------------------------
!
! subroutine for calculation of the porcelli model conditions
!
      SUBROUTINE porcelli_conditions ( issawt, lprint, n_unit &
      , cpf, elongation_at_q1, ion_mass &
      , inductance_at_q1, betapolf1 &
      , beta_i_max, shear_at_q1, r_major, r_minor, rminor_q_1 &
      , grad_norm_press_i_at_q1, denel0, b_toroidal, b_pol_at_q1, ti0 &
      , efast, zfast, pressure_avg_up_to_q1, pressure_at_q1 )
!
! All of the variables in the argument list of this routine are scalars
! that could be computed by the user or by calling sbrtn "porcelli_model".
!
!  issawt = output indicating whether a sawtooth crash should occur
!                  ( issawt=1 --> sawtooth crash,
!                    issawt=0 --> no sawtooth crash)
!
!  lprint  > 0 for diagnostic printout
!  n_unit = output unit number for diagnostic printout
!
!  cpf   = integral of x^(3/2)*p(x) p(x)=ion pressure
!                    (c_p of equ B.7 in reference)
!  elongation_at_q1   = elongation at the q = 1 surface
!  ion_mass   = average ion mass (amu)
!  inductance_at_q1 = internal inductance at q = 1 surface
!  betapolf1   = fast ion beta parameter (equ 27 of reference)
!  beta_i_max   = peak ion toroidal beta
!  shear_at_q1   = rdq/dr at q=1 surface
!  r_major         = major radius (m)
!  r_minor      = minor radius (half-width) at edge of plasma (m)
!  rminor_q_1   = radius of q = 1 surface (m)
!  grad_norm_press_i_at_q1   = |dp_i/dr|/p_i at q=1 surface (m)
!    if p_i/|dp_i/dr|<<<1 take grad_norm_press_at_q1=radius of q=1 surface
!  denel0      = electron density at magnetic axis (10^20 m-3)
!  b_toroidal  = vacuum toroidal field at r_major (testla)
!  b_pol_at_q1 = poloidal field at q = 1 surface (tesla) (outboard midplane)
!  ti0         = central ion temperature (keV)
!  efast       = energy of fast ions at q=1 surface (keV)
!  zfast       = charge of fast ions at q=1 surface
!  pressure_avg_up_to_q1 = volume average pressure within q = 1 surface
!                          (pascal)
!  pressure_at_q1   = pressure at q = 1 surface (pascal)
!
      USE porcelli_constants
!
      USE porcelli_diagnostic_inner
!
!..data module porcelli_diagnostic_inner contains scalar variables
!  that are computed and documented in this routine
!
```

```
      IMPLICIT NONE
!
      INTEGER, INTENT(IN) :: lprint, n_unit
!
      REAL, INTENT(IN) :: cpf
      REAL, INTENT(IN) :: elongation_at_q1, ion_mass &
        , inductance_at_q1 &
        , betapolf1, beta_i_max
      REAL, INTENT(IN) :: shear_at_q1, r_major, r_minor, rminor_q_1 &
       , grad_norm_press_i_at_q1, denel0
      REAL, INTENT(IN) :: b_toroidal, b_pol_at_q1, ti0, efast, zfast &
        ,pressure_avg_up_to_q1, pressure_at_q1
!
      REAL :: zshear_norm  ! mag shear used to normalize delta W
!
      INTEGER, INTENT(OUT) :: issawt
!
```

0.1 Bussac term:

Contributions from ideal MHD Bussac term is $\delta\hat{W}_{\rm Bussac} = -c_{\rm MHD}\epsilon_1^2(\beta_{\rm p1}^2 - \beta_{\rm pc}^2)$ where
$\epsilon_1 = \bar{r}_1/R$ (average q=1 radius/major radius)
$\beta_{\rm p1} = (8\pi/B_{\rm p1}^2)[< p >_1 - p(r_1)]$ (pressure: core + alpha)
$\beta_{\rm pc} = 0.3[1 - (5\bar{r}_1/3\bar{a})]$
$c_{\rm MHD} = \frac{9\pi}{s_1}(l_{\rm i1} - \frac{1}{2})$
$s_1 = \bar{r}_1 q'(\bar{r}_1)$ (magnetic shear parameter)
with <> denoting the volume averaging within q = 1 surface, $\bar{a}$ is average plasma minor radius, and $l_{\rm i1}$ is the internal inductance.

```
!
!
     zshear_norm = sqrt ( shear_at_q1**2 + shear_minimum**2 )
!
     !*********************
     !The Bussac term....delw_bu
     !*********************
!
! zr1avg = average minor radius at q=1
!        = sqrt{ elongation_at_q1 } * rminor_q_1
!
      zr1avg = sqrt( elongation_at_q1 ) * rminor_q_1
!
      zbetap1 = 8.0e-7*zpi &
                 * ( pressure_avg_up_to_q1 - pressure_at_q1 ) &
                 / b_pol_at_q1**2
!
! equ. 12 of Ref
! assuming that the elongation factor cancels out
!
      zbetapc = 0.3 * ( 1.0 - (5.0 * rminor_q_1 ) / ( 3.0 * r_minor ) )
!
! equ. B.4 of Ref
!
      zcmhd = 9.0 * zpi * ( inductance_at_q1 - 0.5 ) &
```

```
              / zshear_norm
!
! equ. B.3 of Ref
!
      zdiffbeta2 = zbetap1**2 - zbetapc**2
!
      zdelwbu = - zcmhd * (zr1avg/r_major)**2 * zdiffbeta2
!
! equ. B.2 in Ref
!
```

0.2 Elongation term

Contributions from the elongation term is $\delta\hat{W}_{\rm el} = -c_{\rm el}[\frac{\kappa_1 - 1}{2}]^2$ where
$c_{\rm el} = \frac{18\pi}{s_1}(l_{\rm i1} - \frac{1}{2})^3$
and $\kappa_1^{1/2}$ is the elongation of the q = 1 surface.

```
!
      !*********************
      !The elongation term....delw_el
      !*********************

      zcel = 18.0 * zpi * ( inductance_at_q1 - 0.5 )**3.0 &
              / zshear_norm
!
! equ. B.6 of Ref
!
      zdelwel = - zcel * ((elongation_at_q1 - 1.0)/2.0)**2
!
! equ. B.5 of Ref
!
      zdelwmhd = zdelwel + zdelwbu
!
! MHD contributions
```

0.3 Kruskal-Oberman term

Contributions from the Kruskal-Oberman term is $\delta\hat{W}_{KO} = 0.6c_{\rm p}\epsilon_1^{1/2}\beta_{\rm i0}/s_1$
with $c_{\rm p} = (5/2)\int_0^1 dx x^{3/2} p_{\rm i}(x)/p_{\rm i0}$, $(x = r/r_1)$
where $p_{\rm i}(x)$ is the ion pressure profile, $p_{\rm i0}$ is its peak value and $\beta_{\rm i0}$ is the peak ion toroidal beta. The expression $\delta\hat{W}_{KO}$ applies to frequency regime $\omega_{*\rm i} < \omega < \omega_{\rm bi}$.

```
!
      !*********************
      !The Kruskal-Oberman term....delw_ko
      !*********************

      zdelwko = 0.6 * cpf * sqrt( zr1avg / r_major ) &
        * beta_i_max / zshear_norm
!
! equ. B.7 of Ref
!
      zdelwc = zdelwmhd+zdelwko ! core contributions
```

0.4 Fast particle term

Contributions from the fast particle term is $\delta\hat{W}_{\rm fast} = c_{\rm f}\epsilon_1^{3/2}\beta^*_{\rm p\alpha}/s_1$ where $\beta^*_{\rm p\alpha} = -\frac{8\pi}{B_{\rm p}^2(r_1)}\int_0^1 dx x^{3/2}\frac{dp}{dx}$, $(x = r/r_1)$ and $c_{\rm f}$ depend depends on the ratio between the mode frequency and the fast particle precessional frequency.

```
!
      !*********************
      !The fast particle term....delw_fast
      !*********************
!
       zdelwfast = cwrat * ( zr1avg / r_major )**1.5 &
         * betapolf1 / zshear_norm
!
! equ. B.8 of Ref
!
       zdelw = zdelwbu + zdelwel + zdelwko + zdelwfast
!
! total contributions
!
      !**************
      ! conditions for sawtooth crash
      !**************

       ztalf = (0.8e-6) * (r_major / b_toroidal ) &
          * sqrt( ion_mass * denel0 )
!
! Alfven time (sec)
!
       zomegf = 500.0 * efast &
        / ( max(zfast,1.0) * b_toroidal * r_major * zr1avg )
!
! fast ion prec. freq. (s^-1)
!
       zomegdi = 1000.0 * ti0 * grad_norm_press_i_at_q1 &
         /( b_toroidal * zr1avg )
!
! ion (z=1) diamagnetic freq. (s^-1)
!
       zrhostar = 3.25e-3 * sqrt( ion_mass * ti0 ) &
         / ( b_toroidal * zr1avg )
!
! ion larmor radius (z=1) to average rminor_q_1
!
       zlundnum = 43.7 * ti0**(1.5) &
         * zr1avg**2 / ztalf
!
! Lundquist number
!
       zdenom = ztalf * zlundnum**(1.0/7.0)
       zgamp  = 1.1 * max ( shear_at_q1, zepslon)**(6.0/7.0) &
         * abs(zrhostar)**(4.0/7.0) / zdenom
!
! semi-collisional m =1 growth rate
```

```
!
       zdelwfast = cwrat &
         * ( zr1avg / r_major )**1.5 &
         * betapolf1 / zshear_norm
!
! equ. B.8 of Ref
!
       eq13L  = -1.0*zdelwc
       eq13R  = chfast*ztalf*zomegf
!
       eq14L  = -1.0*zdelw
       eq14R  = 0.5*zomegdi*ztalf
!
       eq15aL = -1.0*ctrho*zrhostar
       eq15aC = -1.0*zdelw
       eq15aR = 0.5*zomegdi*ztalf
!
       eq15bL = zomegdi
       eq15bR = 1.0*ctstar*zgamp
!
!..First assume that no sawtooth is triggered
!    and then test for each trigger condition
!
       issawt = 0
       leq13  = .FALSE.
       leq14  = .FALSE.
       leq15  = .FALSE.
!
! Test for sawtooth triggered by Eq. 13
!                indicating that fast ion effects are insufficient to
!                stabilize the internal kink mode
!
       IF ( ifastswt > 0) THEN
         IF ( eq13L > eq13R ) THEN
           issawt = 1           ! equ. 13 in Ref, condition 1
           leq13  = .TRUE.
!
           IF ( lprint > 0 ) THEN
             WRITE (n_unit,*) " *******=> sawtooth is triggered by Eq. (13)"
           ENDIF
         ENDIF
         IF ( lprint > 0 ) THEN
           WRITE (n_unit,*) ' Porcelli sawtooth trigger condition Eq. (13)'
           WRITE (n_unit,&
&"(t5,'- delta W_core',t29,'>',t31,'c_h omega_Dh tau_A')")
           WRITE (n_unit,"(1x,3ES25.15)") eq13L, eq13R
           WRITE (n_unit,*)
         ENDIF
       ENDIF
!
! Test for sawtooth triggered by Eq. 14
!                indicating that diamagnetic rotation of thermal ions
!                is insufficient to stabilize the internal kink mode
!
```

```
       IF ( iflarmswt > 0) THEN
         IF ( eq14L > eq14R ) THEN
           issawt = 1           ! equ. 14 in Ref, condition 2
           leq14  = .TRUE.
!
           IF ( lprint > 0 ) THEN
             WRITE (n_unit,*)" ***====> sawtooth is triggered by Eq. (14)"
           ENDIF
         ENDIF
         IF ( lprint > 0 ) THEN
           WRITE (n_unit,*) ' Porcelli sawtooth trigger condition Eq. (14)'
           WRITE (n_unit,&
&"(t5,' - delta W',t29,'>',t31,'0.5 omega_*i tau_A')")
           WRITE (n_unit,"(1x,3ES25.15)") eq14L, eq14R
           WRITE (n_unit,*)
         ENDIF
       ENDIF
!
! Test for sawtooth triggered by Eq. 15
!                indicating that the internal resistive kink mode
!                grows faster than the thermal ion diamagnetic frequency
!
       IF ( ikinswt > 0) THEN
         IF ( eq15aL < eq15aC .and. eq15aC < eq15aR &
                        .and. eq15bL < eq15bR ) THEN
           issawt = 1           ! equ. 15 on Ref, condition 3
           leq15  = .TRUE.
!
           IF ( lprint > 0 ) THEN
             WRITE(n_unit,*)&
 " ***====> sawtooth is triggered by Eq. (15)"
           ENDIF
         ENDIF
         IF ( lprint > 0 ) THEN
           WRITE (n_unit,*) ' Porcelli sawtooth trigger condition Eq. (15)'
           WRITE (n_unit,"(t5,' -c_rho rho',&
               &t29,'<',t31,'- delta W',&
               &t54,'<',t56,'0.5 omega_*i tau_A')")
           WRITE (n_unit,"(1x,3ES25.15)") eq15aL, eq15aC, eq15aR
           WRITE (n_unit,"(t2,'and omega_*i',&
               &t29,'<',t31,'c_* gamma_rho')")
           WRITE (n_unit,"(1x,3ES25.15)") eq15bL,  eq15bR
           WRITE (n_unit,*)
         ENDIF
       ENDIF

       IF ( issawt == 0 .and. lprint > 0 ) then
          WRITE(n_unit,*)&
&" ***====> no sawtooth crash triggered by Porcelli module"
       ENDIF
!
!..diagnostic printout
!
      if ( lprint > 8 ) then
```

```
        WRITE (n_unit,*)
        ! writing the various delw into file
        WRITE (n_unit,110) zdelwbu,zdelwel,zdelwmhd,zdelwko,zdelwc, &
                           zdelwfast,zdelw,ztalf,zomegf,zomegdi,zrhostar, &
                           zr1avg,zlundnum,zgamp,zbetap1
      endif
!
  110  FORMAT("****** ",/, &
              "delw_bu = ",1pe10.3,2x,"delw_el = ",1pe10.3,2x, &
              "delw_mhd = ",1pe10.3, /,"delw_ko = ",1pe10.3,2x, &
              "delw_c = ",1pe10.3,2x,"delw_fast = ",1pe10.3,2x,/, &
              "delw = ",1pe10.3, /,"t_alf(s) = ",1pe10.3,2x, &
              "omeg_f(s^-1) = ",1pe10.3,2x,"omeg_di(s^-1) = ",1pe10.3,/, &
              "rho_star = ",1pe10.3,2x,"r1_avg(m) = ",1pe10.3,2x, &
              "lund_num = ",1pe10.3,2x,/, &
              "semi-coll. m=1 growth rate (s^-1) = ", &
               1pe10.3,2x,"beta_p1 = ",1pe10.3,/, &
              "***************************", /)
!
!
        return
!
        END SUBROUTINE porcelli_conditions
!
!---------------------------------------
!
       END MODULE porcelli_module
```

References

[1] F. Porcelli, D. Boucher, and M. N. Rosenbluth, Plasma Physics and Controlled Fusion, **38**, 2163 (1996).

[2] M. Bussac, R. Pellat, D. Edery, and J. Soule, Phys. Rev. Letters **35**, 1638 (1975).

[3] H. Lutjens, A. Bondeson, and G. Vlad, Nuc. Fusion **32**, 1625 (1992).

[4] M. D. Kruskal and C. R. Oberman, Phys. Fluids **1**, 275 (1958).

[5] D. Campbell *et al.*, Phys. Rev. Letters **60**, 2148 (1988).

[6] F. Porcelli, Plasma Physics and Controlled Fusion **33**, 1601 (1991).

[7] S. Cowley, R. Kulsrud, , and T. Hahm, Phys. Fluids **29**, 3230 (1986).