Supplement of Geosci. Model Dev., 12, 2195–2214, 2019
https://doi.org/10.5194/gmd-12-2195-2019-supplement
© Author(s) 2019. This work is distributed under
the Creative Commons Attribution 4.0 License.

Supplement of

Development and evaluation of pollen source methodologies for the Victorian Grass Pollen Emissions Module VGPEM1.0

Kathryn M. Emmerson et al.

Correspondence to: Kathryn M. Emmerson (kathryn.emmerson@csiro.au)

The copyright of individual parts of the supplement might differ from the CC BY 4.0 License.
1. Larger-scale satellite maps and pasture grass maps

We include larger-scale maps of each pollen count site (Figure S 1), showing the surrounding land use and the spatial variability and fraction of pasture grass. Sites are arranged west to east as described in Table 1 in the main paper.
Figure S 1: larger-scale maps of land use surrounding each pollen count site.

2. Shifted Gaussian distributions based on observed patterns

Figure S 2 shows the observed pollen time series over the 2017 season at each of the count sites. We have fitted a normal distribution to these data, minimising the root mean squared error between the fit and the observations. The mean (mu), standard distribution (sd) and fitting parameter (SF) of each individual fit are given in the top left of each panel.
Figure S 2 Time series of observed pollen at each site, which has had a normal distribution fitted to these data.
3. **Index of Agreement**

The index of agreement (IOA) (Willmott et al. 2012) has been calculated using the ModStats package within Openair (Carslaw and Ropkins, 2012). The description of IOA has been taken from the Openair manual (Carslaw, 2015).

IOA spans between −1 and +1 with values approaching +1 representing better model performance. An IOA of 0.5, for example, indicates that the sum of the error magnitudes is one half of the sum of the observed-deviation magnitudes. When IOA = 0.0, it signifies that the sum of the magnitudes of the errors and the sum of the observed-deviation magnitudes are equivalent. When IOA = −0.5, it indicates that the sum of the error magnitudes is twice the sum of the perfect model-deviation and observed-deviation magnitudes. Values of IOA near −1.0 can mean that the model estimated deviations about O are poor estimates of the observed deviations; however, they also can mean that there is simply little observed variability — therefore, some caution is needed when the IOA approaches −1. It is defined as (with c = 2)

\[
\text{IOA} = \begin{cases} 
1.0 - \frac{\sum |M_i - O_i|}{c \sum |O_i - \bar{O}|}, & \text{when} \sum |M_i - O_i| \leq c \sum |O_i - \bar{O}| \\
\frac{c \sum |O_i - \bar{O}|}{\sum |M_i - O_i|} - 1.0, & \text{when} \sum |M_i - O_i| > c \sum |O_i - \bar{O}| 
\end{cases}
\]

4. **Gerrity score**

For a multi-category forecast we may wish to reward correct forecasts of rare events more than correct forecasts of common events, and penalise forecasts that are very wrong more than forecasts that are only a slightly wrong. A class of scores called "Gerrity scores" (GS) do just this by multiplying the contingency table by a scoring matrix (which is really a set of weights) and summing to get a score. That is,

<table>
<thead>
<tr>
<th>Observed</th>
<th>Scoring matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>L</td>
<td>n_{11}</td>
</tr>
</tbody>
</table>
GS scores measure the skill, relative to random chance, of predicting the correct category and ranges from -1 to 1 (perfect). It can be used with multi-category forecasts of any dimension, but is probably most often applied to 3-category forecasts. The scoring matrix is computed from the climatology (base rate) of the observations, i.e. how often each category occurs, and has nothing to do with the forecasts. One can either use the long-term climatology or the sample climatology. The long-term climatology is better if it is available, but it is often the case that it must be estimated from the sample.

The following are the steps for computing the elements of the scoring matrix:

1. If using the sample climatology, compute the observed relative probability of each category \( i = 1, 2, 3 \) as 
   \[
   p_i = \frac{n_i}{N}
   \]
   where \( n_i \) is the total number observed in each category and \( N \) is the total number of samples.

2. Compute intermediate variables \( a_i \) from the relative probabilities,
   \[
   a_1 = (1 - p_1) / p_1 \quad a_2 = (1 - p_1 - p_2) / (p_1 + p_2) \quad a_3 = (1 - p_1 - p_2 - p_3) / (p_1 + p_2 + p_3).
   \]

3. Compute diagonal elements of the scoring matrix as
   \[
   s_{11} = 0.5 \times (a_1 + a_2) \\
   s_{22} = 0.5 \times \left( \frac{1}{a_1} + a_2 \right) \\
   s_{33} = 0.5 \times \left( \frac{1}{a_1} + \frac{1}{a_2} \right)
   \]

4. Compute off-diagonal elements of the scoring matrix as
   \[
   s_{12} = s_{21} = 0.5 \times (a_2 - 1) \\
   s_{13} = s_{31} = -1 \\
   s_{23} = s_{32} = 0.5 \times \left( \frac{1}{a_1} - 1 \right)
   \]

For the Melbourne pollen forecasts, the scoring matrix is

\[
\begin{pmatrix}
0.24 & -0.45 & -1.00 \\
-0.45 & 1.38 & 0.83 \\
-1.00 & 0.83 & 6.00
\end{pmatrix}
\]
When multiplied by the three-category contingency table on the first page we get a Gerrity score for the 1-day forecasts of GS=0.468. It is greater than zero and therefore is a skilful value.

More information about verification scores is available online at www.cawcr.gov.au/projects/verification.

5. Wind roses

We plot wind roses (Figure S 3) for each of the AWS sites listed in Table 1 of the main paper during the period from 1 October 2017 to 31 December 2017.

Hamilton Airport

Ballarat aerodrome (Creswick)
6. References


7. Pollen emissions code VGPEM1.0

SUBROUTINE CTM_pollen_emissions (nx, ny, nz, zfull, dxy, waspd, U10m, V10m, sfctemp, sfcpres, mix_ratio, prcpR, prcpU, tscr_24h, SH_24h, CTM_clock_current_LST, geoGrid, zenith, pollen_veg, evi_gradient, last_available, EVIfallDaySmoothed, EVIwinterMaxSmoothed, Srate, PollenFluxes)

IMPLICIT NONE

INTEGER, INTENT(IN) :: nx  !east/west points
INTEGER, INTENT(IN) :: ny  !north/south points
INTEGER, INTENT(IN) :: nz  !vertical points
REAL, INTENT(IN), DIMENSION(nx,ny,nz) :: zfull  !unscaled cell heights (m)
REAL, INTENT(IN), DIMENSION(nx,ny) :: dxy        !cell areas (m2) - unscaled
REAL, INTENT(IN), DIMENSION(nx,ny) :: wspd      !wind speed (m/s)
REAL, INTENT(IN), DIMENSION(nx,ny) :: U10m      !U wind speed (m/s)
REAL, INTENT(IN), DIMENSION(nx,ny) :: V10m      !V wind speed (m/s)
REAL, INTENT(IN), DIMENSION(nx,ny) :: sfctemp   !surface temp (K)
REAL, INTENT(IN), DIMENSION(nx,ny) :: sfcpres      !surface pres (Pa)
REAL, INTENT(IN), DIMENSION(nx,ny,nz) :: mix_ratio   !mixing ratio (g/g)
REAL, INTENT(IN), DIMENSION(nx,ny) :: prcpR  !resolved precipitation (m/s)
REAL, INTENT(IN), DIMENSION(nx,ny) :: prcpU  !convective precipitation (m/s)

TYPE(clock_variable) :: CTM_clock_current_LST   !CTM current time in local solar time

TYPE(geo), INTENT(IN), DIMENSION(nx,ny) :: geoGrid !lat/long of all grid points
REAL, INTENT(IN), DIMENSION(nx,ny) :: zenith     !zenith angle by grid point
REAL, INTENT(IN), DIMENSION(nx,ny) :: pollen_veg  !veg for pollen
REAL, INTENT(IN), DIMENSION(nx,ny) :: evi_gradient   !Gradient of the EVI
REAL, INTENT(INOUT), DIMENSION(nx,ny) :: last_available ! last available pollen on plants
REAL, INTENT(INOUT), DIMENSION(nx,ny,1) :: EVIfallDaySmoothed
REAL, INTENT(INOUT), DIMENSION(nx,ny,1) :: EVIwinterMaxSmoothed

REAL, INTENT(INOUT) :: Srate(nx*ny,number_species) !Emsn rate array!cumulative emission rates (g/m3/s)

REAL :: pollen_EF  ! emission factor for pollen based on jday
REAL :: relhum  ! relative humidity (0-1, just calculated again)
REAL :: relhum_24h  !24h relative humidity
real :: local_e
real :: local_solar_time !the local solar time 0.0 to 23.99
real :: logit_y1 = 0.05, logit_y2 = 0.95 ! ordinates for the *_x1 and *_x2 values
real :: prc_x1, prc_x2  ! abcissae where logistic function reaches logit_y1, logit_y2 for precip
real :: prc_alpha, prc_c  ! logistic function parameters for precip
real :: prc_logit0       ! ordinate of precip logistic function at prc=0
real :: rh_x1,  rh_x2   ! abcissae where logistic function reaches logit_y1, logit_y2 for rel hum
real :: rh_alpha,  rh_c  ! logistic function parameters for rel hum
real :: loc_solar_mu    ! timing of the mean daily pollen release (in fractional hours)
real :: loc_solar_sigma1 ! standard deviation around the mean daily pollen release (in fractional hours)
real :: loc_solar_sigma2
real :: precip_sum_target ! rainfall amount [cm] required to achieve the maximum capacity in terms of pollen emissions
real :: doy_mu    ! mean day-of-year (in fractional days) of the pollen emissions peak
real :: doy_sigma ! standard deviation (in fractional days) around the emissions peak
real :: soy_mu    ! mean second-of-year (in seconds) of the pollen emissions peak
real :: soy_sigma ! standard deviation (in seconds) around the emissions peak
real :: immediate_timing  ! scaling factor for the immediate emissions (between 0 and 1). Essentially the fraction to release.
real :: gross_timing ! scaling factor for the emissions based on phenology, this varies slowly with time. It take values between 0 and 1. Essentially the fraction to release.
real :: spatial ! the spatial scaling factor
real :: rh_factor
real :: prc_factor
real :: f_wind
real :: hour_factor1, hour_factor2
real :: t2_factor
real :: prc
real :: loss_factor       ! fraction of pollen available lost from plant to wet and dry deposition [0, 1]
real :: available       ! available pollen mass to release
real :: prod           ! amount of pollen produced per timestep
real :: loss           ! amount of pollen lost per timestep
real :: released       ! amount of pollen emitted per timestep
real :: rhour          ! the model time in UTC [fractional hour, i.e. 0 to 23.999]
real :: capacity_limit ! maximum theoretical production [g/m2]
real, parameter :: sqrt_two_pi = 2.506628274631 ! sqrt(2*pi)
real, parameter :: KtoC = 273.15     ! 0 degrees C in degrees Kelvin
real, parameter :: f_stagnant = 0.33 ! suppression factor in stagnant conditions
real, parameter :: f_promote  = 0.67 ! promotion factor under windy conditions
real, parameter :: usatur     = 5.0  ! saturation wind speed [m/s]
real, parameter :: tsatur     = 15.0 ! saturation temperature
real, parameter :: T_mid  = 15.0 ! midpoint of temperature factor [C]
real, parameter :: T_rate = 0.33 ! rate of change in temperature factor at midpoint [1/C], before scaling factor is applied
! maximum theoretical production
real :: max_capacity ! [g/m2]
logical :: first
real, parameter :: dtstep = 3600.0 ! seconds since last function call (1 hour)

real :: yday ! day of year
real, DIMENSION(nx,ny) :: prcpThisDay, prcpPreviousDay ! current & previous days precipitation in units of millimeters
! local variables
real :: a0
real :: b0
real :: mu
real :: ydayTest(3)
integer :: iMinYdayTest
real :: ydayToUse
real, parameter :: sigma = 19.

! 1 kg / hectare = 1000 g / (10000 m^2) = 1 g / (10 m^2) = 0.1 g m^-2
! 464 kg/hectare = maximum theoretical pollen emission for perennial rye-grass (Smart, Tuddenham, Knox; 1979; Aust. J.
Bot. ; V27, pp. 333-342)
max_capacity = 46.4 ! [g/m2]

! Need to convert LST hour into a local time (+10 for Victoria).
time_zone=150./15.

IF (CTM_clock_current_LST%hr .LE. (23.-time_zone)) THEN
  local_solar_time=CTM_clock_current_LST%hr + time_zone
ELSE
  local_solar_time=CTM_clock_current_LST%hr - (time_zone + 1)
END IF
yday = real(CTM_clock_current_LST%jd)

! Initialise flux area
PollenFluxes(:,:,)%Poln_1=0.0
Pfx(:,:,)=0.0

! Apply normal distribution with integrated total = 464 kg/hectare as per Smart et al 1979
! depends on julian day, and season being oct 1st (jd 273, day 1) to dec 31st (day 92)
! ignore leap year. From C:\Thunderstorm Asthma\Unit tracer results\equation of normal distribution

kday=CTM_clock_current_LST%jd - 273  ! this becomes x
pollen_EF=exp(-0.5*((kday-46.5)/26.702)**2))*9.53E-8 ! adjusted for 464 kg hec-1

! define parameters for the pollen emissions functions (logistic representation)

! relative humidity
rh_x1 = 0.5
rh_x2 = 0.8
rh_c = (rh_x1 + rh_x2)/2.0
rh_alpha = 2 * log((1 - logit_y1)/logit_y1)/(rh_x2 - rh_x1)

! precipitation -
prc_x1 = 0.0
prc_x2 = 0.5 ! originally in units of cm
prc_c = (prc_x1 + prc_x2)/2.0
prc_alpha = 2 * log((1 - logit_y1)/logit_y1)/(prc_x2 - prc_x1)

loc_solar_mu = 12.0 ! Smart & Knox, Aust. J. Bot. 1979, 27, 317-331, figure 6 KME - bimodal?
loc_solar_sigma1 = 2.0 ! Smart & Knox, Aust. J. Bot. 1979, 27, 317-331, figure 6
loc_solar_sigma2 = 4.0
! this is normalised to take value 1.0 when loc_solar_time == loc_solar_mu
hour_factor1 = gaussian_pdf(local_solar_time, loc_solar_mu, loc_solar_sigma1) * (loc_solar_sigma1 * sqrt_two_pi)
hour_factor2 = gaussian_pdf(local_solar_time, loc_solar_mu, loc_solar_sigma2) * (loc_solar_sigma2 * sqrt_two_pi)

doy_mu = 319.0 ! de Morton et al., Int J Biometeorol (2011) 55:613-622, fig 1 - KME: SAYS NOV 30th = 334
doy_sigma = 18.0 ! de Morton et al., Int J Biometeorol (2011) 55:613-622, fig 1
soy_mu = doy_mu *24.0*60.0*60.0 ! convert from day-of-year to second-of-year
soy_sigma = doy_sigma*24.0*60.0*60.0 ! convert from day-of-year to second-of-year

first = .true.
Pfx(i,j)=0.
! fill pointer area in Srate with PollenFlux
DO k=1,number_pollen_sizes
! fill PollenFlux array
DO j=1,ny
DO i=1,nx

oneD=i+(j-1)*nx !nxny

relhum = calc_rel_humid(sfctemp(i,j), sfcpres(i,j), mix_ratio(i,j,1) )
relhum_24h = calc_rel_humid(tscr_24h(i,j), sfcpres(i,j), SH_24h(i,j) )
prc = prcpR(i,j) + prcpU(i,j) * 3.6e6 !m/s ->mm/hr *1000*3600

! emissions decrease with relative humidity (hence negative rh_alpha)
rh_factor = f_stagnant + f_promote * logistic(relhum, rh_c, -rh_alpha) ! moderated
! rh_factor = logistic(relhum, rh_c, -rh_alpha)

! emissions decrease with precipitation (hence negative prc_alpha), also normalise so that no penalty is applied for zero rainfall
prc_factor = f_stagnant + f_promote * logistic(prc, prc_c, -prc_alpha)/prc_logit0 ! moderated
! prc_factor = logistic(prc, prc_c, -prc_alpha)/prc_logit0

! temperature factor (follows the wind speed factor)
t2_factor = f_stagnant + f_promote * logistic(sfctemp(i,j) - KtoC, T_mid, T_rate)

! following equation 11 in Sofiev, 2013
f_wind = f_stagnant + f_promote * (1 - exp(-(wspd(i,j))/usatur))

SELECT CASE (pollen_emissions_parameterisation)
case (1)
  immediate_timing = wspd(i,j)
  gross_timing = pollen_EF
  spatial = pollen_veg(i,j)
  Pfx(i,j) = immediate_timing * gross_timing * spatial

case (2)
  immediate_timing = wspd(i,j)
  gross_timing = evi_gradient(i,j)
  spatial = pollen_veg(i,j)
  Pfx(i,j) = immediate_timing * gross_timing * spatial

case (3)
  immediate_timing = wspd(i,j)
  gross_timing = evi_gradient(i,j)
  spatial = 1.0 ! embodied in evi_gradient
  Pfx(i,j) = immediate_timing * gross_timing * spatial

case (4)
  immediate_timing = hour_factor1 * rh_factor * prc_factor * f_wind * t2_factor
  gross_timing = evi_gradient(i,j)
  spatial = pollen_veg(i,j)
  Pfx(i,j) = immediate_timing * gross_timing * spatial

case (5)
  immediate_timing = hour_factor1 * rh_factor * prc_factor * f_wind * t2_factor
  gross_timing = pollen_EF
  spatial = pollen_veg(i,j)
\[ Pfx(i,j) = \text{immediate\_timing} \times \text{gross\_timing} \times \text{spatial} \]

\text{case (6)}
\[ \text{immediate\_timing} = \text{hour\_factor2} \times \text{rh\_factor} \times \text{prc\_factor} \times f\_\text{wind} \times t2\_\text{factor} \]
\[ \text{gross\_timing} = \text{pollen\_EF} \]
\[ \text{spatial} = \text{pollen\_veg(i,j)} \]
\[ Pfx(i,j) = \text{immediate\_timing} \times \text{gross\_timing} \times \text{spatial} \]

\text{case (7)}
\[ ! \text{Production-loss model.} \]
\[ ! \]
\[ ! \text{emis} = \text{available} \times \text{immediate\_timing} \]
\[ ! \text{available} = \text{last\_available} + \text{production} - \text{loss} \]
\[ ! \text{loss} = L(\text{last\_available}, \text{rain}, \text{humidity}) \]
\[ ! \text{production} = \text{capacity} \times \text{gross\_timing} \]
\[ ! \text{capacity} = f(\text{landuse}, \text{prec\_sum}) \]
\[ ! \text{gross\_timing} = g(\text{temp\_sum}, \text{lat}) \]
\[ ! \text{immediate\_timing} = h(\text{wind}, \text{humidity}, \text{time\_of\_day}) \]
\[ ! \]
\[ \text{gross\_timing} = \text{pollen\_EF} \]
\[ \text{immediate\_timing} = \text{hour\_factor2} \times \text{rh\_factor} \times \text{prc\_factor} \times f\_\text{wind} \times t2\_\text{factor} \]
\[ ! \text{prc has units mm/hr} \]
\[ \text{loss\_factor} = \text{calculate\_loss\_factor}(\text{prc}, \text{dtstep}) \]
\[ \text{capacity\_limit} = \text{max\_capacity} \]
\[ \text{prod} = \text{capacity\_limit} \times \text{pollen\_veg(i,j)} \times \text{gross\_timing} \times \text{dtstep} \]
\[ \text{loss} = \text{last\_available( i,j )} \times \text{loss\_factor} \]
\[ \text{available} = \text{last\_available( i,j )} + \text{prod} - \text{loss} \]
\[ \text{released} = \text{available} \times \text{immediate\_timing} \]
\[ Pfx( i,j ) = \text{released} \]
\[ ! \text{Note: we don't subtract the loss term ...} \]
\[ \text{last\_available(i,j)} = \text{available} - \text{released} ! \] because it's equivalent to: \[ \text{last\_available(i,j)} = \text{last\_available(i,j)} + \text{prod} - \text{loss} - \text{released} \]

\text{case (8)}
\[ ! \text{shifted Gaussian shapes depending on lon,lat} \]
immediate_timing = hour_factor2 * rh_factor * prc_factor * f_wind * t2_factor

IF (geoGrid(i,j)%lat .GT. -37.0) THEN
  pollen_EF = exp(-0.5*(((kday-34.65)/15.45)**2))*9.53E-8 ! Bendigo and Dookie
ELSEIF (geoGrid(i,j)%lat .LE. -37.0 .AND. geoGrid(i,j)%lon .GT. 143.5) THEN
  pollen_EF = exp(-0.5*(((kday-50.5)/19.3)**2))*1.2E-7 ! Creswick and Churchill
ELSE
  pollen_EF = exp(-0.5*(((kday-48.1)/7.7)**2))*1.56E-7 ! Hamilton
ENDIF

gross_timing = pollen_EF
spatial = pollen_veg(i,j)
Pfx(i,j) = immediate_timing * gross_timing * spatial

case (9)
  prcpThisDay(i,j) = prc *24. ! should be the past 24-hours rainfall [mm]
  prcpPreviousDay(i,j) = prc *24. ! should be the previous 24 hours of rainfall [mm]

  Pfx( i,j ) = emitAlaStatisticalModel_v1(EVIfallDaySmoothed(i,j,1), EVIwinterMaxSmoothed(i,j,1),
                                         prcpThisDay(i,j), prcpPreviousDay(i,j), sfctemp(i,j) - KtoC, relhum * 100., U10m(i,j), V10m(i,j), yday)

case (10)
  prcpThisDay(i,j) = prc*24. ! should be the past 24-hours rainfall [mm]
  prcpPreviousDay(i,j) = prc*24. ! should be the previous 24 hours of rainfall [mm]

  Pfx( i,j ) = emitAlaStatisticalModel_v2(EVIfallDaySmoothed(i,j,1), EVIwinterMaxSmoothed(i,j,1),
                                         prcpThisDay(i,j), prcpPreviousDay(i,j), sfctemp(i,j) - KtoC, relhum * 100., U10m(i,j), V10m(i,j), yday)

case default
  Pfx( i,j ) = 0.0
end SELECT
Srate(oneD,pPN(k))=Srate(oneD,pPN(k))+Pfx(i,j)

!This allows more than one size fraction within poln_1.
! also change the units here from g/m3/s to g/cell area/s while in nxny loop

5  PollenFluxes(oneD,k)%poln_1 = Pfx(i,j)/zfull(i,j,1)*dxy(i,j)

END DO !ny
END DO !nx

10  END DO !pollen size pointer

END SUBROUTINE CTM_pollen_emissions

MODULE CTM_pollen_auxiliary_routines

contains

!-----------------------------------------------------------------------
!  calculate relative humidity based on water vapour mixing ratio, temperature & pressure
!-----------------------------------------------------------------------
real function calc_rel_humid(ta, prs, qv)

  implicit none
  real :: calc_rel_humid  ! relative humidity, OUTPUT, units = fraction (0-1)
  real, intent(in) :: ta  ! air temperature,    INPUT, units = K
  real, intent(in) :: prs ! pressure,           INPUT, units = PA
  real, intent(in) :: qv  ! water vapor mixing ratio, INPUT, units = kg/kg

  ! constants
  real, parameter  :: vp0   = 611.29 ! vapor press of water at 0 C [ Pa ]
  real, parameter  :: svpt0 = 273.15 ! constant for saturation vapor pressure [K]
  real, parameter  :: mvoma = 0.622  ! ratio of mol. weight of water vapor to mol weight of air [dimensionless]

  ! local variables
  real :: e_sat ! sat vap pres (Pa) as fn of T (deg K)
  real :: tempc ! air temperature, units = C
  real :: qsat  ! sat water vapor mixing ratio, units = kg/kg
  real :: h2ovp ! ambient water vapor pressure [ Pa ]
tempc = ta - svpt0

e_sat = vp0 * exp( 17.625 * tempc / ( 243.04 + tempc ) )

!   qsat = e_sat * mvoma / ( prs - e_sat )

5       rh = qv / qsat

h2ovp = prs * qv / ( mvoma + qv )
calc_rel_humid = MAX( 0.005, MIN( 0.99, h2ovp / e_sat ) )

return

datafunction calc_rel_humid

!----------------------------------------------------------------------
! Logistic function (a smooth approximation to a piecewise-linear ramp between 0 and 1)
!----------------------------------------------------------------------

function logistic(x, c, alpha) result(y)

implicit none

real, intent(in) :: x       ! abcissa of the logistic function
real, intent(in) :: c       ! midpoint (abcissa when ordinate = 0.5)
real, intent(in) :: alpha   ! rate of increase (dydx(x=c) = 0.25 alpha)
real             :: y       ! ordinate

y = 1.0/(1.0 + exp(-alpha*(x-c)))

return

datafunction logistic

!----------------------------------------------------------------------
! A smooth function that approximates y = min(x,1.0)
!----------------------------------------------------------------------

function smooth_ramp(x) result(y)

implicit none

! arguments
real, intent(in) :: x
real             :: y

y = 1.0/(1.0 + exp(-alpha*(x-c)))

return

datafunction smooth_ramp
! local parameters
real, parameter :: alpha = 10.0 ! smoothness parameter. Increasing alpha means a sharper transition from y=x to y=1.0
real, parameter :: c = 0.999995459903963 ! intercept, \( c = \log(\exp(\alpha) - 1)/\alpha \), so that smooth_ramp(0.0) = 0.0

\[
y = 1 - \frac{1}{\alpha}\log(1 + \exp(\alpha(c - x)))
\]

! relationship between alpha and c
! 0 = 1 - \frac{1}{\alpha}\log(1 + \exp(\alpha c))
! 1 = \frac{1}{\alpha}\log(1 + \exp(\alpha c))
! alpha = \log(1 + \exp(\alpha c))
! \exp(\alpha) = 1 + \exp(\alpha c)
! \exp(\alpha c) = \exp(\alpha) - 1
! alpha c = \log(\exp(\alpha) - 1)
! c = \log(\exp(\alpha) - 1)/\alpha

return
end function smooth_ramp

!-----------------------------------------------------------------------
! The Gaussian probability density function
!-----------------------------------------------------------------------
function gaussian_pdf(x, mu, sigma) result(fx)
  implicit none
  real, intent(in) :: x ! the abcissa of the Gaussian distribution PDF
  real, intent(in) :: mu ! the mean of the Gaussian distribution
  real, intent(in) :: sigma ! the standard deviation of the Gaussian distribution
  real :: fx ! the ordinate of the Gaussian distribution PDF

! parameters
real, parameter :: sqrt_two_pi = 2.506628274631

fx = exp\left( -0.5 \times \frac{(x - mu)^2}{sigma^2} \right) / (sigma \times sqrt_two_pi)

return
end function gaussian_pdf
! Calculate the loss rate based on exponential decay, accelerated by rainfall
!

function calculate_loss_factor(rain, tstep) result(loss_factor)

  implicit none
  real :: loss_factor ! output: fraction of pollen available lost from plant to wet and dry deposition [0, 1]
  real, intent(in) :: rain ! rain per hour [cm]
  real, intent(in) :: tstep ! length of timestep [s]

  real, parameter :: half_life_dry = 2.0 ! pollen assumed to have a half-life of 2 days on the plant in dry conditions
  real, parameter :: half_life_rain = 0.5 ! pollen assumed to have a half-life of 0.5 days on the plant in rainy conditions
  real, parameter :: rain_thresh = 0.2 ! threshold rain rate per hour above which the pollen half-life is set to half_life_rain [cm]
  real, parameter :: ln2 = 0.693147180559945

  real :: half_life
  real :: rain_factor
  real :: lambda ! decay rate

  real :: dt

  real, parameter :: seconds_per_day = 86400.0

  rain_factor = min(rain/rain_thresh,1.0)
  half_life = rain_factor*half_life_rain + (1.0-rain_factor)*half_life_dry
  lambda = ln2/half_life
  dt = tstep/seconds_per_day
  loss_factor = dt*lambda

  ! rain = 0.0 => rain_factor = 0.0 => half_life = half_life_dry
  ! rain = rain_thresh => rain_factor = 1.0 => half_life = half_life_rain
  ! rain = 0.5*rain_thresh => rain_factor = 0.5 => half_life = 0.5*(half_life_rain + half_life_dry)

  return

end function calculate_loss_factor
! Estimate the seasonal pollen capacity based on the rainfall

function capacity_factor(precip_sum, max_capacity, precip_sum_target) result(capacity)
implicit none
real, intent(in) :: precip_sum
real, intent(in) :: max_capacity
real, intent(in) :: precip_sum_target
real :: capacity

capacity = max_capacity * smooth_ramp(precip_sum/precip_sum_target)

return
end function capacity_factor

! probability density function of the Cauchy distribution
function dcauchy(x, x0, gamma) result (y)
 implicit none
! arguments
real, intent(in) :: x ! x-value at which to calculate the
real, intent(in) :: x0 ! location parameter for the distribution
real, intent(in) :: gamma ! spread parameter for the distribution
real :: y ! the resulting PDF value
! local variables
real, parameter :: pi = 3.14159265358979

y = ( 1.0 / (pi * gamma)) * (gamma**2 / ( (x - x0)**2 + gamma**2 ) )

return
end function dcauchy

! copied from numerical recipes in F77 (SUBROUTINE LOCATE)
subroutine find_interval(n,xvals,xtest,idx)
 implicit none
! arguments
integer, intent(in) :: n
real, intent(in) :: xvals(n), xtest
integer, intent(out) :: idx

! local variables
integer :: ilow, imid, iup

ilow = 0
iup = n+1

10 10 if(iup-ilow .gt. 1) then ! If we are not yet done,
imid=(iup+ilow)/2 ! compute a midpoint,
if((xvals(n) .ge. xvals(1)) .eqv. (xtest .ge. xvals(imid))) then
   ilow = imid ! and replace either the lower limit
else
   iup = imid ! or the upper limit, as appropriate.
endif
   goto 10 ! Repeat until
endif ! test condition 10 is satisfied.

20 if(xtest .eq. xvals(1)) then ! Then set the output
   idx = 1
else if(xtest.eq.xvals(n))then
   idx = n-1
else
   idx = ilow
endif
return
end subroutine find_interval

function emitAlaStatisticalModel_v1(EVIfallDaySmoothed, EVIwinterMaxSmoothed, &
   prcpThisDay, prcpPreviousDay, TM, RH, U10m, V10m, yday,) result(emis)
   implicit none
! arguments
real, intent(in) :: EVIfallDaySmoothed

real, intent(in) :: EVIwinterMaxSmoothed
real, intent(in) :: prcpThisDay
real, intent(in) :: prcpPreviousDay
real, intent(in) :: TM
real, intent(in) :: RH
real, intent(in) :: U10m, V10m
real, intent(in) :: yday
real :: emis

! local variables
real :: a0
real :: b0
real :: mu

real :: a1
real :: b1
real :: sf
real :: lnsmooth
real :: l1pRN, l1pRNm1upwind
real :: WD, wd2
real :: f5x5, f6x6
integer :: idx
real :: lnFC
real :: TMtest, RHtest
real, parameter :: sigma = 19.
real :: ydayTest(3)
integer :: iMinYdayTest
real :: ydayToUse

! regression coefficients from the EVI fall day to pollen peak day
! MU = (a0 + b0 * EVIfallDaySmoothed(ix,iy)) %% 365
based on unsmoothed data:

\[ a_0 = 246.679166 \]
\[ b_0 = 0.251085 \]

based on smoothed data:

\[ a_0 = 0 \]
\[ b_0 = 1 \]
\[ \mu = \text{mod}(a_0 + b_0 \times \text{EVI}_{\text{fall day smooth}}, 365.0) \]

regression coefficients from the EVI winter max to the pollen season magnitude

\[ \text{SF} = \text{pmax}(a_1 + b_1 \times \text{EVI}_{\text{winter max smooth}}(ix,iy),0) \]

based on smoothed data:

\[ a_1 = -4355.913 \]
\[ b_1 = 21490.343 \]
\[ \text{sf} = \text{max}(a_1 + b_1 \times \text{EVI}_{\text{winter max smooth}},0.0) \]

The following chunk of code determines whether the centre of the phenology curve is closest to the day-of-year \( y_{\text{day}} - 365 \), \( y_{\text{day}} \) or \( y_{\text{day}} + 365 \). The \( y_{\text{dayToUse}} \) is then the value among \( y_{\text{day}} - 365, y_{\text{day}} \) or \( y_{\text{day}} + 365 \) that is closest to day-of-year \( '\mu' \) (the centre of the distribution).

\[ \text{ydayTest} = (/ \text{abs}(y_{\text{day}} - 365 - \mu), \text{abs}(y_{\text{day}} - \mu), \text{abs}(y_{\text{day}} + 365 - \mu) /) \]
\[ \text{iMinYdayTest} = \text{minloc}(\text{ydayTest},1) \]
\[ \text{ydayToUse} = y_{\text{day}} + 365 \times \text{real}(\text{iMinYdayTest} - 2) \]

\[ \ln\text{smooth} = \log(\text{dcauchy}(y_{\text{dayToUse}},\mu,\sigma) \times \text{sf}) \]

prcpThisDay should be in units of mm

\[ \ln p_{\text{RN}} = \log(1.0 + \text{prcpThisDay}) \]

prcpPreviousDay should be in units of mm

\[ \ln p_{\text{RN}+1\text{upwind}} = \log(1.0 + \text{prcpPreviousDay}) \]

\[ \text{TMtest} = \max(\min(\text{TM}, \text{TM}_{x}(\text{nTMpoints})), \text{TM}_{x}(1)) \]
\[ \text{call find_interval(nTMpoints,TM}_{x},\text{TMtest},\text{idx}) \]
\[ f5x5 = \text{TM}_{y}(\text{idx}) \]
RHtest = max(min(RH, RH_x(nRHpoints)), RH_x(1))
call find_interval(nRHpoints,RH_x,RHtest,idx)
f6x6 = RH_y(idx)

\[ \text{lnFC} = -0.290379277 + 0.970041571 \times \text{lnsmooth} - 0.182604732 \times 11\text{pRNm1upwind} + \& \\
& -0.117335246 \times 11\text{pRN} + f5x5 + f6x6 \]
emis = exp(lnFC)-1.0

return
end function emitAlaStatisticalModel_v1

function emitAlaStatisticalModel_v2(EVIfallDaySmoothed, EVIwinterMaxSmoothed, &
    prcpThisDay, prcpPreviousDay, TM, RH, U10m, V10m, yday) result(emis)

    implicit none
    ! arguments
    real, intent(in) :: EVIfallDaySmoothed
    real, intent(in) :: EVIwinterMaxSmoothed
    real, intent(in) :: prcpThisDay
    real, intent(in) :: prcpPreviousDay
    real, intent(in) :: TM
    real, intent(in) :: RH
    real, intent(in) :: U10m, V10m
    real, intent(in) :: yday
    real :: emis

    ! local variables
    real :: a0
    real :: b0
    real :: mu
    real :: a1
    real :: b1
    real :: sf
real :: lnsmooth
real :: llpRN,

real :: WD, wd2
integer :: idx
real :: lnFC
real :: TMtest, RHtest
real, parameter :: sigma = 19.

real :: ydayTest(3)
integer :: iMinYdayTest
real :: ydayToUse

real :: RH2test, sRH2val
real :: TM4test, sTM4val
real :: RNtest, sRNval

real :: WS3

! regression coefficients from the EVI fall day to pollen peak day
! MU = (a0 + b0 * EVIfallDaySmoothed(ix,iy)) \% 365
! ! based on smoothed data:
a0 = 181. + 21.477641

b0 = 0.384697
mu = mod(a0 + b0 * EVIfallDaySmoothed, 365.0)

! regression coefficients from the EVI winter max to the pollen season magnitude
! SF = pmax(a1 + b1 * EVIwinterMaxSmoothed(ix,iy),0)
! ! based on smoothed data:
a1 = 267.6271

b1 = 8853.9903
sf = min(max(a1 + b1 * EVIwinterMaxSmoothed, 0.0), 1.5*6894.355)

! The following chunk of code determines whether the centre of
! the phenology curve is closest to the day-of-year yday-365,
! yday or yday+365. The ydayToUse is then the value among
! yday-365, yday or yday+365 that is closest to day-of-year 'mu'
! (the centre of the distribution).

5 ydayTest = (/ abs(yday - 365 - mu), abs(yday - mu), abs(yday + 365 - mu) /)
iMinYdayTest = minloc(ydayTest,1)
ydayToUse = yday + 365.*real(iMinYdayTest - 2)

Insmooth = log(dcauchy(ydayToUse,mu,sigma)*sf)

WS3 = sqrt(U10m**2 + V10m**2)

RH2test = max(min(RH, RH2_x(nRH2points)), RH2_x(1))
call find_interval(nRH2points,RH2_x,RH2test,idx)
sRH2val = RH2_y(idx)

TM4test = max(min(TM, TM4_x(nTM4points)), TM4_x(1))
call find_interval(nTM4points,TM4_x,TM4test,idx)
sTM4val = TM4_y(idx)

RNtest = max(min(prcpThisDay, RN_x(nRNpoints)), RN_x(1))
call find_interval(nRNpoints,RN_x,RNtest,idx)
sRNval = RN_y(idx)

25 lnFC = 1.225480027 + 0.769707046 * Insmooth - 0.032793100 * WS3 + sRH2val + sTM4val + sRNval

emis = exp(lnFC)-1.0

return

30 end function emitAlaStatisticalModel_v2

END MODULE CTM_pollen_auxiliary_routines