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

Kathryn M. Emmerson[1], Jeremy D. Silver[2], Edward, Newbigin[3], Edwin R. Lampugnani[3], Cenk Suphioglu[4], Alan Wain[5] and Elizabeth Ebert[5]

5   [1]Climate Science Centre, CSIRO Oceans & Atmosphere, Aspendale, VIC 3195. Australia
[2]School of Earth Sciences, University of Melbourne, VIC 3010 Australia.
[3]School of BioSciences, University of Melbourne, VIC 3010, Australia
[4]School of Life and Environmental Sciences, Deakin University, Waurn Ponds, VIC 3216, Australia.
[5]Bureau of Meteorology, Docklands, VIC 3008, Australia.

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

1.   Shifted Gaussian distributions based on observed patterns

Figure S 1 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.
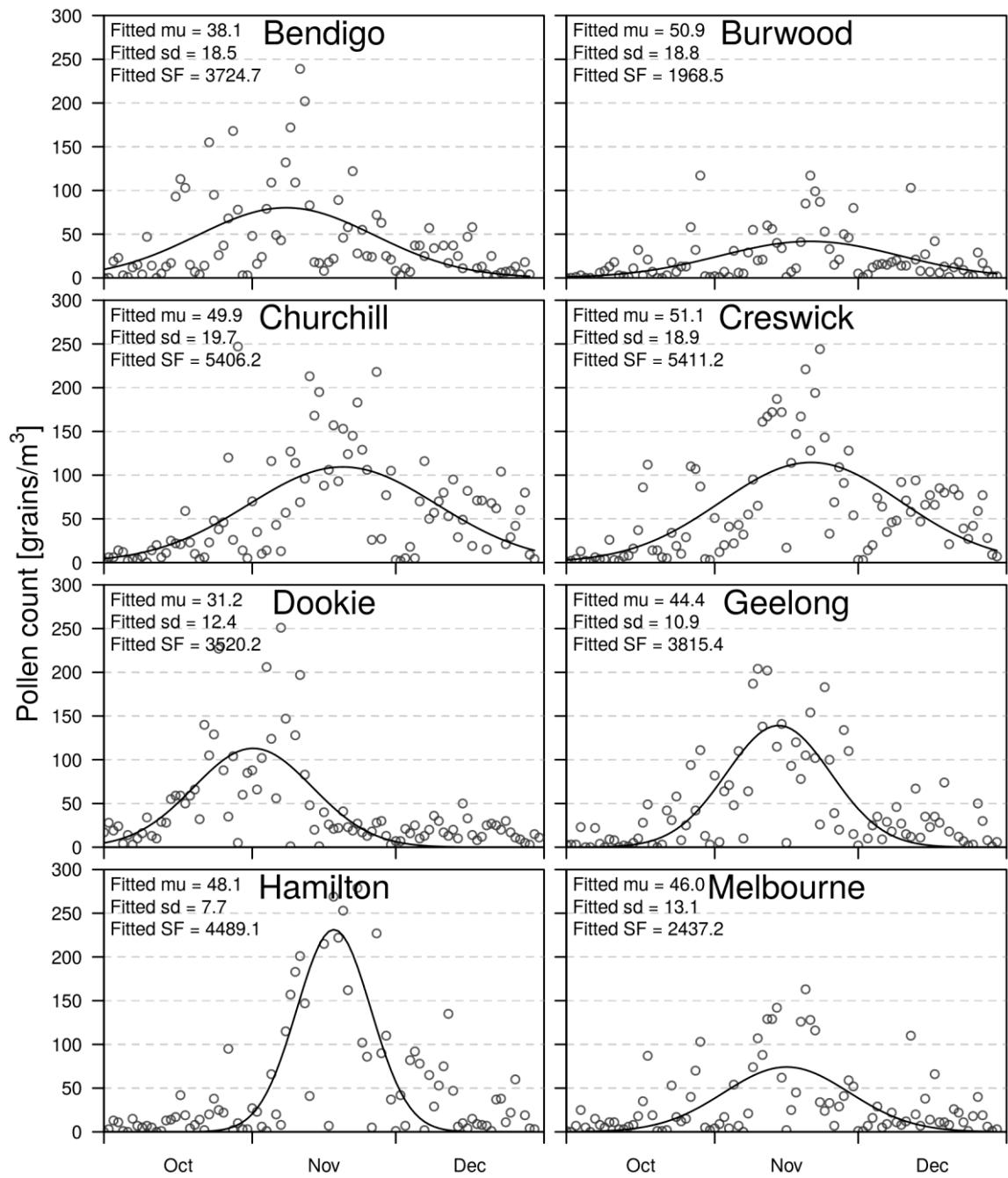
Figure S 1 Time series of observed pollen at each site, which has had a normal distribution fitted to these data.

2. 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; but, they also can mean that there simply is little observed variability — so some caution is needed when the IOA approaches −1. It is defined as (with c = 2):

$$
IOA = \begin{cases} 1.0 - \dfrac{\sum_{i=1}^{n} |M_i - O_i|}{c \sum_{i=1}^{n} |O_i - \overline{O}|}, \text{ when } \sum_{i=1}^{n} |M_i - O_i| \leq c \sum_{i=1}^{n} |O_i - \overline{O}| \\[2em] \dfrac{c \sum_{i=1}^{n} |O_i - \overline{O}|}{\sum_{i=1}^{n} |M_i - O_i|} - 1.0, \text{ when } \sum_{i=1}^{n} |M_i - O_i| > c \sum_{i=1}^{n} |O_i - \overline{O}| \end{cases}
$$

3. 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 badly wrong more than forecasts that are only a little bit 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,

	Observed				Scoring matrix		
	L	M	H+				
L	n_{11}	n_{12}	n_{13}		s_{11}	s_{12}	s_{13}

Forecast	M	n_{21}	n_{22}	n_{23}	**X**	s_{21}	s_{22}	s_{23}
	H+	n_{31}	n_{32}	n_{33}		s_{31}	s_{32}	s_{33}
	total	n_1	n_2	n_3				

$$GS = \frac{1}{N}\sum_{i=1}^{3}\sum_{j=1}^{3} n_{ij} s_{ij}$$

GS measures 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

5   has nothing to do with the forecasts. You can either use the long-term climatology or the sample climatology. The long-term climatology is better if you have it but it is often the case that you have to estimate it from the sample.

Here 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 $p_i = n_i / N$ where

10   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,

$$a_1 = (1 - p_1)/p_1 \qquad a_2 = (1 - p_1 - p_2)/(p_1 + p_2) \qquad a_3 = (1 - p_1 - p_2 - p_3)/(p_1 + p_2 + p_3)$$

15   3. Compute diagonal elements of the scoring matrix as

$$s_{11} = 0.5 * (a_1 + a_2)$$
$$s_{22} = 0.5 * \left(\frac{1}{a_1} + a_2\right)$$
$$s_{33} = 0.5 * \left(\frac{1}{a_1} + \frac{1}{a_2}\right)$$

4. Compute off-diagonal elements of the scoring matrix as

$$s_{12} = s_{21} = 0.5 * (a_2 - 1)$$
$$s_{13} = s_{31} = -1$$
$$s_{23} = s_{32} = 0.5 * \left(\frac{1}{a_1} - 1\right)$$

For the Melbourne pollen forecasts, the scoring matrix is $S =$

0.24	- 0.45	- 1.00
- 0.45	1.38	0.83
- 1.00	0.83	6.00

When multiplied by the 3-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!

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

4. References

Carslaw, D.C. and K. Ropkins, (2012). openair — an R package for air quality data analysis. Environmental Modelling & Software. Volume 27-28, pp. 52–61.

Carslaw, D.C. (2015). The openair manual — open-source tools for analysing air pollution data. Manual for version 1.1-4,
10   King's College London.

Willmott, C. J., Robeson, S. M. and Matsuura, K. (2012), A refined index of model performance. Int. J. Climatol., 32: 2088-2094. doi:10.1002/joc.2419.

5. Pollen emissions code VGPEM1.0

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

IMPLICIT NONE

20   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
25   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)
30   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
 5    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
10    REAL, INTENT(IN), DIMENSION(nx,ny,1) :: EVIfallDaySmoothed
      REAL, INTENT(IN), DIMENSION(nx,ny,1) :: EVIwinterMaxSmoothed

       REAL, INTENT(INOUT) :: Srate(nx*ny,number_species) !Emsn rate array!cumulative emission rates (g/m3/s)
      TYPE(PollenFlux), INTENT(INOUT),DIMENSION(nx*ny,number_pollen_sizes) :: PollenFluxes   !pollen flux in g/m3/s
15
     ! local definitions
      INTEGER :: i,j,k,oneD,kday
      REAL, DIMENSION(nx,ny) ::Pfx !holding area
      REAL :: pollen_EF  ! emission factor for pollen based on jday
20    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
25    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
30    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
```

```fortran
    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 phenlology, 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]
    ! parameters
    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 preciption in units of millimeters
        ! local variables
        real :: a0
   5    real :: b0
        real :: mu
        real :: ydayTest(3)
        integer :: iMinYdayTest
        real :: ydayToUse
  10    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.
  15    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.
  20    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
  25
        yday = real(CTM_clock_current_LST%jd)

        !

  30    !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
5      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
10     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)

15     prc_x1 = 0.0
       ! precipitation -
       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)
20     prc_logit0 = logistic(0.0, prc_c, -prc_alpha)
       precip_sum_target = 40.0 ! de Morton et al., Int J Biometeorol (2011) 55:613-622, fig 4

       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
25     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)

30
       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
```

```fortran
       first = .true.
       Pfx(i,j)=0.
5    ! fill pointer area in Srate with PollenFlux
       DO k=1,number_pollen_sizes

     ! fill PollenFlux array
       DO j=1,ny
10       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) )
15       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)
20       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
25       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)
30
         ! following equation 11 in Sofiev, 2013
         f_wind = f_stagnant + f_promote * (1 - exp(-(wspd(i,j))/usatur))

         SELECT CASE (pollen_emissions_parameterisation)
```

```fortran
        case (1)
            immediate_timing = wspd(i,j)
            gross_timing = pollen_EF
            spatial = pollen_veg(i,j)
5           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)
10          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
15          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)
20          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)
25          Pfx(i,j) = immediate_timing * gross_timing * spatial
        case (6)
            immediate_timing = hour_factor2 * rh_factor * prc_factor * f_wind * t2_factor
            gross_timing = pollen_EF
            spatial = pollen_veg(i,j)
30          Pfx(i,j) = immediate_timing * gross_timing * spatial

        case (7)
            ! Production-loss model.
            !
```

```
                    ! emis = available * immediate_timing
                    ! available = last_available + production - loss
                    ! loss = L(last_available, rain, humidity)
                    ! production = capacity * gross_timing
  5                 ! capacity = f'(landuse, prec_sum)
                    ! gross_timing = g(temp_sum, lat)
                    ! immediate_timing = h(wind, humidity, time_of_day)
                    !
                    gross_timing = pollen_EF
 10                 immediate_timing = hour_factor2 * rh_factor * prc_factor * f_wind * t2_factor
                    ! prc has units mm/hr
                    loss_factor = calculate_loss_factor(prc , dtstep)
                    capacity_limit = max_capacity
                    prod = capacity_limit * pollen_veg(i,j) * gross_timing * dtstep
 15                 loss = last_available( i,j ) * loss_factor
                    available = last_available( i,j ) + prod - loss
                    released = available * immediate_timing
                    Pfx( i,j ) = released
             ! Note: we don't subtract the loss term ...
 20                 last_available(i,j) = available - released ! .. because it's equivalent to: last_available(i,j) = last_available(i,j) + prod -
             loss - released

                 case (8)
                     !shifted Gaussian shapes depending on lon,lat
 25                  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

 30                 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)
5            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]
10
                 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)
15        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,)
20
         case default
           Pfx( i,j ) = 0.0
           end SELECT

25      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
       PollenFluxes(oneD,k)%poln_1 = Pfx(i,j)/zfull(i,j,1)*dxy(i,j)
30
      END DO !ny
     END DO !nx

     END DO !pollen size pointer
```

END SUBROUTINE CTM_pollen_emissions

MODULE CTM_pollen_auxiliary_routines

contains

```
    !---------------------------------------------------------------------
    !  calculate relative humdity based on water vapour mixing ratio, temperature & pressure
    !---------------------------------------------------------------------
10  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
15    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]
20    ! 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 ]
25
      tempc = ta-svpt0
      e_sat = vp0 * exp( 17.625 * tempc / ( 243.04 + tempc ) )
      !      qsat = e_sat * mvoma / ( prs - e_sat )
      !      rh = qv / qsat
30    h2ovp = prs * qv / ( mvoma  + qv )
      calc_rel_humid = MAX( 0.005, MIN( 0.99, h2ovp / e_sat ) )

      return
     end function calc_rel_humid
```

```fortran
      !---------------------------------------------------------------------
      !      Logistic function (a smooth approximation to a piecewise-linear ramp between 0 and 1)
      !---------------------------------------------------------------------
5     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)
10      real          :: y      ! ordinate

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

        return
15    end function logistic

      !---------------------------------------------------------------------
      !      A smooth function that approximates y = min(x,1.0)
      !---------------------------------------------------------------------
20    function smooth_ramp(x) result(y)
        implicit none
        ! arguments
        real, intent(in) :: x
        real          :: y
25      ! 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 - (1/alpha)*log(1 + exp(alpha*(c - x)))
30
        ! relationship between alpha and c
        !      0 = 1 - (1/alpha)*log(1 + exp(alpha*c))
        !      1 = (1/alpha)*log(1 + exp(alpha*(c))
        !      alpha = log(1 + exp(alpha*c))
```

```fortran
    !        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( -0.5 * (((x-mu)/sigma)**2)) / (sigma * 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]
```

```fortran
      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)
```

```fortran
      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    if(iup-ilow .gt. 1)then ! If we are not yet done,
        imid=(iup+ilow)/2 ! compute a midpoint,
```

```fortran
            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.
5           endif
            goto 10 ! Repeat until
          endif ! test condition 10 is satisfied.

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

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

        ! local variables
        real :: a0
```

```
          real :: b0
          real :: mu

          real :: a1
    5     real :: b1
          real :: sf

          real :: lnsmooth
          real :: l1pRN, l1pRNm1upwind
   10
          real :: WD, wd2
          real :: f5x5, f6x6
          integer :: idx
          real :: lnFC
   15     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:
   25     ! a0 = 246.679166
          ! b0 = 0.251085
          ! ! based on smoothed data:
          a0 = 0.0
          b0 = 1.0
   30     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 = -4355.913
```

```
          b1 = 21490.343
          sf = max(a1 + b1 * EVIwinterMaxSmoothed,0.0)

          ! The following chunk of code determines whether the centre of
5         ! 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).
          ydayTest = (/ abs(yday - 365 - mu), abs(yday - mu), abs(yday + 365 - mu) /)
10        iMinYdayTest = minloc(ydayTest,1)
          ydayToUse = yday + 365.*real(iMinYdayTest - 2)

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

15        ! prcpThisDay should be in units of mm
          l1pRN = log(1.0+prcpThisDay)

          ! prcpPreviousDay should be in units of mm
          l1pRNm1upwind = log(1.0+prcpPreviousDay)
20
          TMtest = max(min(TM, TM_x(nTMpoints)), TM_x(1))
          call find_interval(nTMpoints,TM_x,TMtest,idx)
          f5x5 = TM_y(idx)

25        RHtest = max(min(RH, RH_x(nRHpoints)), RH_x(1))
          call find_interval(nRHpoints,RH_x,RHtest,idx)
          f6x6 = RH_y(idx)

            lnFC = -0.290379277 + 0.970041571 * lnsmooth - 0.182604732 * l1pRNm1upwind + &
30          & -0.117335246 * l1pRN + f5x5 + f6x6
          emis = exp(lnFC)-1.0

           return
          end function emitAlaStatisticalModel_v1
```

```fortran
      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 :: l1pRN,

      real :: WD, wd2
      integer :: idx
      real :: lnFC
      real :: TMtest, RHtest
      real, parameter :: sigma = 19.
      real :: ydayTest(3)
      integer :: iMinYdayTest
```

```fortran
      real :: ydayToUse

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

      real :: WS3

10    ! 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
15    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
20    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,
25    ! 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).
      ydayTest = (/ abs(yday - 365 - mu), abs(yday - mu), abs(yday + 365 - mu) /)
      iMinYdayTest = minloc(ydayTest,1)
30    ydayToUse = yday + 365.*real(iMinYdayTest - 2)

      lnsmooth = 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)

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

        emis = exp(lnFC)-1.0

        return
      end function emitAlaStatisticalModel_v2

    END MODULE CTM_pollen_auxiliary_routines
```