RIDEV V2, Rice Model of Phenology and Thermal Sterility of Spikelets

Michael DINGKUHN, Richard Pasco, Cecile JULIA and Jean-Christophe SOULIE

CIRAD/IRRI

This document describes RIDEV, a crop model simulating rice phenology and thermal-stress related spikelet sterility, based on the end-of-project report to AfricaRice (2012) which funded development of this model. The scientific basis and data are described in Julia and Dingkuhn (2012, 2013), and in more detail in the Ph.D. thesis of Cecile Julia, funded jointly by the RISOCAS project (GTZ), CIRAD and AfricaRice.

1. Scientific background and objectives

1.1 The old RIDEV (V1)

We describe here a new version of RIDEV, a cereal model of crop phenology and thermal stresses causing sterility of the spikelets on inflorescences, causing yield losses. The original RIDEV model, developed in 1995 for irrigated rice systems in the Sahel (Dingkuhn, 1997; Wopereis et al., 2003), was implemented in GW Basic language and had two versions, RIDEV_Sim (enabling predictive simulations for climatic risk analyses and farmers' guidance to chose appropriate crop calendars) and RIDEV_Cal (enabling parameterization against target files containing phenological observations, using a simple optimization procedure). The model was initially used for a regional study of thermal risks (Dingkuhn et al., 1995), varietal characterization (Dingkuhn and Miezan, 1995) and elaboration of cropping calendars (Dingkuhn, 1995), and thereafter as an agronomic decision tool by WARDA (now AfricaRice) and its NARS partners.

RIDEV differs from other crop models (1) in its simplicity, (2) focus on phenology and sterility, and (3) consideration of crop microclimate, namely simulation of organ temperature as a function of the apex' position in the soil-floodwater-atmosphere continuum.
The original RIDEV model increasingly became obsolete for a couple of reasons:

- The software basis (GW Basic on DOS) is outdated
- No investment in software maintenance and improvement
- Changes in germplasm, for which no calibration was available
- Apparently, changes in crop phenology at the Senegal study sites, either through genetic drift in the varieties (analogous to sorghum and millet during the same period? (…)) or climate change (CC), resulting in loss of model accuracy with the old parameter settings.

1.2 Need for a new RIDEV

A new version of RIDEV is needed, based on modern programming language, expanded domain of geographic validity and incorporation of new scientific knowledge on rice biology. The model is needed for the following applications:

- Simulator of the crop time frame of a new decision toll for farmers currently under development at AfricaRice and IRRI, the Rice Manager (predictive mode of model application)
- Applications in the context of climate risk analyses for crops, measurement of CC impacts and development of adaptation strategies for CC, as pursued by CCAFS, GRiSP and other CGIAR Research Programs (predictive mode)
- Model-assisted phenotyping of photothermal response traits in rice and cereals in general, through parameter optimization against observed datasets, as required by the GRiSP Global Rice Phenotyping Network (…) and the ORYTAGE project (reverse or heuristic mode)

The first two applications require a modernized equivalent of RIDEV_Sim, but will also require an equivalent of RIDEV_Cal as parameterization tool. The last application (phenotyping) mainly requires a new equivalent of RIDEV_Cal equipped with a powerful parameter optimization tool suited to multi-environment fitting for hundreds of genotypes (context of genetic QTL and association studies).

The main potential users are AfricaRice, IRRI and CCAFS, as well as their local, regional and global collaborators.

1.3 Skills expected from the new RIDEV (V.2)

The following skills are expected of RIDEV V.2:

- Simulation of the date of the main phenological events panicle initiation (PI), flowering (F) and maturity (M) of the crop as a function of genotype, sowing date, cultural practice (flooded-irrigated or upland; transplanting or direct seeding), thermal conditions of the apical growing point and day length dynamics (function of calendar date and geographical latitude)
- Simulation of fraction of sterile spikelets resulting from adverse temperatures (heat or cold) of the affected organs during their sensitive periods (before and at flowering)
- Achievement of such simulations with minimal data requirements to enable applications at sites where environmental data are scarce: daily minimum and maximum air temperature (Tmin, Tmax), geo-reference, and if necessary, some commonly available atmospheric variables from weather stations (but not solar radiation, which is frequently unavailable)
- Achievement of model calibration for specific genotypes using easily measureable phenological variables and spikelet sterility fraction (but this for a variety of sowing dates and/or latitudes as source of thermal and day-length variation)
- An in-built tool for parameter optimization using automated sensitivity analysis, suited for multi-environment fitting of parameters and serial parameterization of large genotype numbers as needed for model-assisted phenotyping

The specific skill of the original RIDEV, namely to simulate key components of microclimate (e.g., floodwater temperature) are to be conserved and improved in RIDEV V.2. In addition, heat-induced spikelet sterility is to be simulated on the basis of two innovations, (1) time of day of anthesis and (2) panicle temperature at that time. These innovative improvements benefit from the results of Cecile Julia's thesis on exactly those issues (Chapters 1 and 2).

Lastly, RIDEV V.2 will also optionally simulate phenology under dryland conditions (but without considering drought), in a mode that simply bypasses the calculation of the microclimate specific to flooded irrigated rice (use of air temperature at 2m instead of simulated microclimate).

Description of RIDEV V.2 model

1.4 Simulation of thermal time

Daily heat units (degree-days) are calculated from Tmin and Tmax of the appropriate entity (air, floodwater, a weighted average of both, or panicle, depending on developmental stage) using a linear broken-stick model. A physiological temperature Tphys is calculated as state variable. Temperatures below Tbase are disregarded (Tphys=0), temperatures between Tbase and Topt are considered effective (Tphys=Tact-Tbase), and temperatures above Topt are also considered ineffective (Tphys=Topt).

To account for diurnal patterns of T, which potentially incur Tact<Tbase or Tact>topt during part of the day), an empirical standard distribution of T during the 24-h cycle is implemented with hourly integration. This relative diurnal pattern is dimensioned on a daily basis using T extrema (Tmin and Tmax of the entity concerned) and calculated day length including civil twilight (the night and day portions of the pattern are "stretched" or "compressed" according to photoperiod). A partial Tphys is calculated for each hour and accrued for 24 hours.

Thermal time of a day is equal to Tphys of the same day.

1.5 Simulation of phenological phases

RIDEV V.2 considers sequentially a Basic Vegetative Phase (BVP), a Photoperiod-Sensitive Phase (PSP), a Reproductive phase (RPR) and a maturation phase (MAT). BVP, RPR and MAT are considered to be of constant thermal duration for a given genotype, unless transplanting shock as parameterized by the user extends the BVP. PSP thermal duration is considered to depend on both temperature and day length on the basis of 4 alternative models the user can choose from.

Thermal time (or a day-length dependent term) is accrued until a critical value is attained, followed by the onset of the subsequent phonological phase.

1.6 Simulation of photoperiod sensitivity

Because considerable uncertainty persists on the way how day length affects PSP duration, and because different cereal species behave differently, the user is given the choice of 4 models of photoperiod sensitivity.

PP Model 1: Linear "quantitative" model

This model follows the classical concept of Major and also resembles the model implemented in the original RIDEV. It accrues [dPsp = PPsens * Tphys * max(PP-PPcrit, 0)] using [Psp = Psp + dPsp] until [PSP >= 1], which is when panicle initiation (PI) happens and the RPR starts. PPsens sets the sensitivity to day length (0…1) and PPcrit (10…12) sets the day length below which no response to it occurs. The model assumes a gradual (quantitative) response, as opposed to a threshold-type (qualitative) response.

PP Model 2: Curvi-linear model able to approximate both qualitative and quantitative responses

This model is identical to Model 1 except for the introduction of parameter PPexp (1…10), an exponent that renders the effect of day length progressive. This is found in many traditional cereals, notably in sorghum and millet. If [PPexp=1] then Model 2 gives the same results as Model 1.

PP Model 3: Effect of day-to-day change of day length

This model was considered by Clerget et al. (2004) for sorghum and supposes that short-day plants are actually decreasing-day plants. The model calculates at day-to-day change in day length [dPP = PP-PPold] and accrues the term [dPsp = PPsens * Tphys * max(dPP-PPcrit, 0)]. PPsens can assume values [0…1] and PPcrit [0…0.2], the latter usually with very small values (0.003 or less). This model does not always give convincing results but was considered here because it makes a fundamentally different assumption on photoperiodism.

PP Model 4 : Impatience

This model was specifically developed for sorghum where it is the only one able to accurately predict the crop's phenology in the off-season (Dingkuhn et al., 2008). Its broader applicability for crops remains to be evaluated but there are indications that it captures a generic phenomenon. The principle is that PI occurs when the plant is exposed to a critical day length (shortness of day), but that this threshold moves as the plant's wait state (PSP) gets longer. This is the "impatience" principle, analogous to animal ethology (threshold lowering under prolonged wait states for an external signal). PPsens=0…1.5, PPexp=0.1…1, SeuilPP=12.5…15. (PPcrit, a parameter also present in this model, was permanently set to 11.5 because it seems not to vary.)

1.7 Simulation of floodwater temperature

Floodwater temperature can be very different from air temperature through evaporative cooling and shading by the canopy. We used data generated by the thesis of Cecile Julia from Senegal (cold and hot seasons), Camargue (summer) and IRRI (dry season) to construct an empirical model. An initial attempt to use a multiple regression model was abandoned because it gave unrealistic predictions for extremely humid situations (characterized by small daily Tmax-Tmin differentials) that were not represented in the data but do occur in the monsoonal wet season. We thus constructed a logical model that forced T(water)-T(air) differentials to be equal to zero when air Tmax-Tmin equals zero (theoretical humid situation with minimal sunlight).

The resulting model is presented in the box below. Its parameters were optimized against observations using the solver utility of Microsoft Excel.

Empirical, optimized models to calculate water T from Tmax and Tmax-Tmin of the air as well as the crop light transmission ratio (LTR) as calculated from LAI:

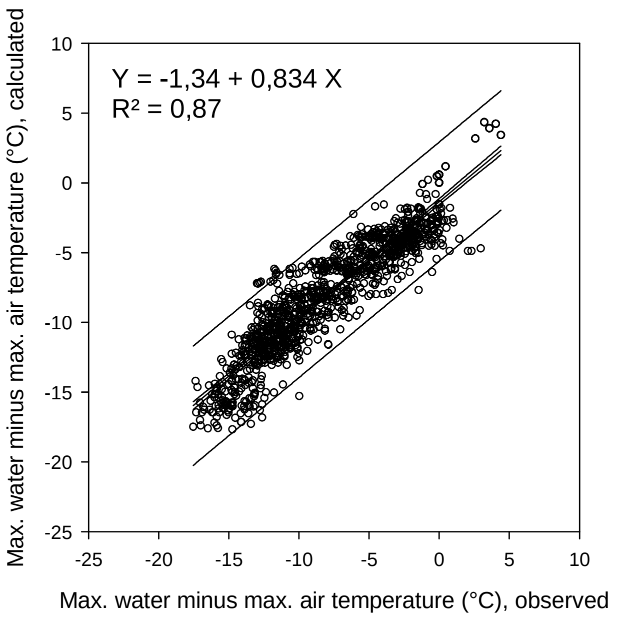
Model TWmax:

TWmax-Tmax = [a(Tmax-Tmin)*(1-LTR) + b(Tmax-Tmin)*LTR] * (Tmax+c)/20
therefore:
TWmax = Tmax + [a(Tmax-Tmin)*(1-LTR) + b(Tmax-Tmin)*LTR] * (Tmax+c)/20
A=-0,728; b=0,275; c=-18,46

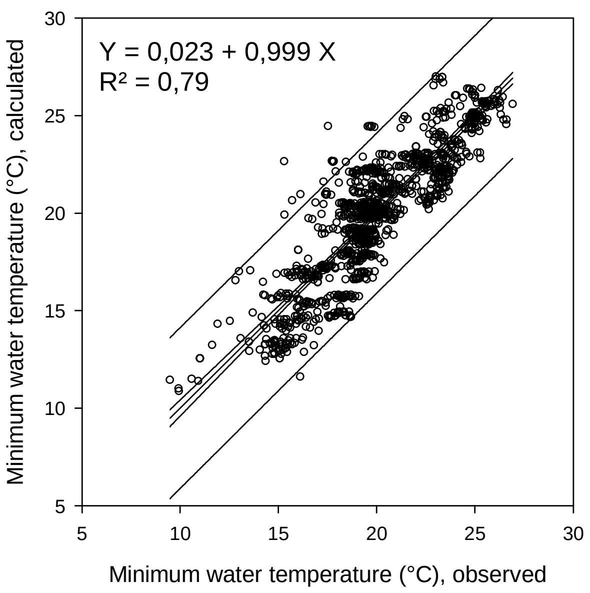
Model TWmin:

TWmin = Tmin + a(Tmax-Tmin)*(1-LTR) + b(Tmax-Tmin)*LTR + c((Tmax-Tmin)^1.2)
A=0,658; 0,425; c=-0,303

These models gave the simulated vs observed correlations shown in the Figure 1 below.



Model:
TWmax-Tmax = [a(Tmax-Tmin)*(1-LTR) + b(Tmax-Tmin)*LTR] * (Tmax+c)/20
TWmax = Tmax + [a(Tmax-Tmin)*(1-LTR) + b(Tmax-Tmin)*LTR] * (Tmax+c)/20
a =-0.728, b = 0,275, c = -18,46



Model:
TWmin = Tmin + a(Tmax-Tmin)*(1-LTR) + b(Tmax-Tmin)*LTR + c((Tmax-Tmin)^1.2)
a = 0,658, b = 0,425, c = 0,303

<u>Figure. 1</u>. Correlations between calculated and observed water temperatures. Top : daily max and min differential ; bottom : minimum temperature.

The models require validation with independent data but they are likely to be robust because (1) they are based on a large range of climatic situations and more than 1000 observations, and (2) they force T(water)-T(air) differentials towards zero for humid and low light situations, which is true for theoretical considerations.

TWmax varied between 18 ^{o} below Tmax (on hot, dry days under a dense crop cover) and 5 ^{o} above Tmax (on hot, dry days in the absence of a crop cover). TWmin was on average similar to Tmin of the air because it varied by a few degrees in both directions (not presented).

1.8 Simulation of apex temperature of the plant

Under irrigated conditions, it was assumed that Tapex = Twater until PI. Thereafter, the apex rises above the floodwater due to stem elongation and gradually approximates the air temperature. This was simulated by using weighted averages between Twater and Tair, evolving from PI (Tapex=Twater) to flowering (Tapex=Tair).

For upland, non-irrigated condition, we assumed Tapex=Tair. This is a simplification and probably erroneous, but it may serve as such as a reference for the approach used by other crop models, which do not consider microclimate.

1.9 Simulation of panicle temperature

For model development for panicle temperature, data was available for the cold and hot seasons in Senegal, based on thermal images (thesis Cecile Julia). Corresponding data for IRRI (Philippines) were not yet available and will be used for model validation.

The following model predicting the panicle-air temperature differential (TD) was determined by multiple linear regression analysis:

<u>Table 1</u>: Description of the model explaining the air and panicle temperature difference TD: Results of the Multiple linear regression of TD against 5 parameters (air temperature, relative humidity and solar radiation at 2m, the panicle relative position in the canopy and the light transmission ratio); and results of the analysis of variance. The equation was found using data from Senegal only (cold and hot dry seasons).

TD = 0,782 + 0,422 Ta_2m - 0,0443 RH_2m - 0,00287 Rs - 8,05 Z - 6,59 LTR

With :

Ta_2m = T(air) at 2m

RH_2m = rel. air humidity at 2m

Rs = solar radiation (W/sqm)

Z = mean panicle position relative to canopy height (fraction, 0.7…1)

LTR= canopy light transmission ratio (0…1 theoretically, but mostly 1)

Multiple linear regression :

	Coeff	Stdev	Std Coef	t-ratio	P
Constant	7,82E-01	5,56E-01	0,00E+00	1,40E+00	8,01E-02
Ta_2m	4,22E-01	1,04E-02	6,29E-01	4,04E+01	0,00E+00
RH_2m	-4,43E-02	2,50E-03	-2,99E-01	-1,77E+01	0,00E+00
Rs	-2,87E-03	1,77E-04	-1,37E-01	-1,62E+01	0,00E+00
Z	-8,05E+00	3,38E-01	-1,94E-01	-2,38E+01	0,00E+00
LTR	-6,59E+00	4,49E-01	-1,31E-01	-1,47E+01	0,00E+00

R2 = 0,819   R2 ajusted = 0,818

Analysis of variance

	DF	SS	MS	F	P
Regression	5,00	19493,33	3898,67	2613,61	0,00^{E}+00
Residual error	2896,00	4319,90	1,49		
Total	2901,00	23813,23			

The following Figure presents graphically the fit of this model:

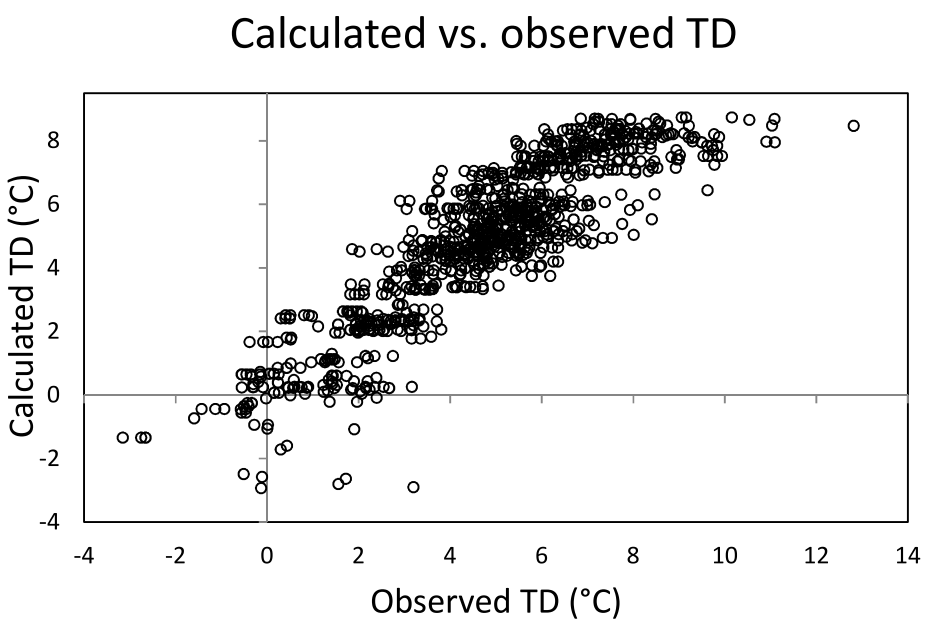


Figure. 2. Correlation between simulated and observed panicle temperature

We used this relationship in the RIDEV V.2 model to calculate the daily maximal panicle temperature TPANmax) from canopy properties and weather conditions at the hottest time of the day. For the minimum panicle temperature we assumed equilibrium with the air (TPanMin = Tmin) because transpiration can assumed to be negligible at night and Rs is nil.

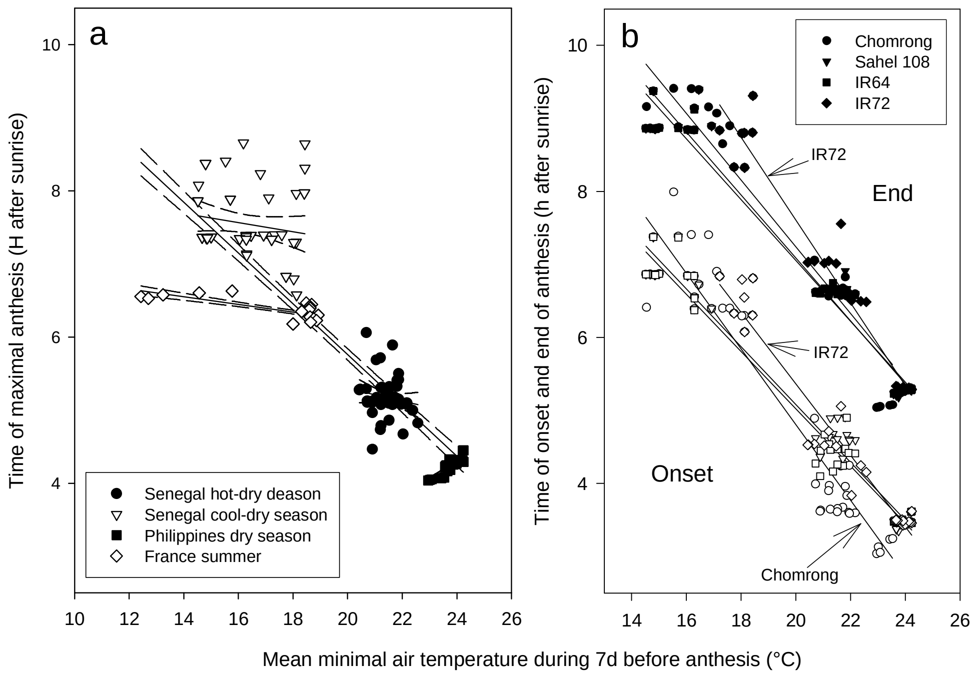
The following model was thus implemented, which includes an estimate of incident global radiation at midday (W/sqm) from Rs (MJ):

```
RsMax = 22 // at 12h day length

WattPerSqm = 1000 * (Rs * 12 / daylength) / RsMax

TPanMax = Tmax – (0.78 + 0.422 * Tmax – 0.0443 * RHmin – 0.00287 * WattPerSqm – 8.05 * Z – 6.59 *
LTR)

TPanMin = Tmin
```

Panicle temperature at any given time of day was estimated by applying TPANmax and TPANmin to a standard diurnal, relative temperature pattern as described further below.

1.10 Simulation of time of day of anthesis and panicle temperature at anthesis

Combined results from Senegal, France and the Philippines showed that environmental effects on time of day of anthesis are large whereas varietal differences were small in the study (Fig. 3). High night temperatures (Tmin) advanced anthesis, providing potentially for escape from heat stress. High RH also advanced anthesis (not shown), potentially providing for escape from heat stress associated with poor transpirational cooling under humid conditions. An article was submitted to Eur. J. Agronomy on these results and accepted pending minor revision.



<u>Figure. 3</u>. Relationship of time of anthesis vs. minimum air temperature (Tmin). <u>Fig. 1a:</u> Maximal anthesis vs. Tmin. Correlation across 4 environments, Y=12.7-0.348X with R^2=0.80; within environments, not significant. <u>Fig.1b:</u> Onset and end of anthesis vs. Tmin, by genotype across 3 tropical environments (Senegal and Philippines). Correlations for onset of anthesis: Chomrong, Y=15.2-0.517X, R^2=0.90; Sahel 108, Y=13.0-0.396X, R^2=0.96; IR64, Y=12.9-0.392X, R^2=0.97; IR72, Y=15.2-0.489X, R^2=0.94. Correlations for end of anthesis: Chromrong, Y=16.4-0.456X, R^2=0.88; Sahel 108, Y=15.7-0.431X, R^2=0.95; IR64, Y=15.4-0.416X, R^2=0.97; IR72, Y=18.9-0.567X, R^2=0.95. Correlations marked by arrows are significantly ($P<0.05$) different from those of other genotypes.

We only considered the thermal effect on anthesis time in RIDEV V.2. Since duration of anthesis was constant at about 2h, we simulate only the time of mid anthesis using the results shown in Fig. 3b, and express it as the number of hours after sunrise (hasr). A crop parameter sets the time of anthesis under conditions of [Tmin=20^{o}C] expressed in hours before noon (HrAnthBefNoon20C). The following equation is implemented in RIDEV V.2:

AnthesisTime = (14 – 0.4 * Tmin) + (24-PP)/2 – HrAnthBefNoon20C

Consequently, if the genotypic parameter [HrAnthBefNoon20C=0h] and [Tmin=20^{o}C] then anthesis centers at midday (11h-13h). If [Tmin = 25^{o}C] then anthesis centers at 10h, etc.

For the time of anthesis, the panicle temperature is estimated on the basis of a diurnal simulation of panicle temperature, using a relative diurnal template (explained further below) and the simulated TPANmax and TPANmin of the day.

1.11 Simulation of diurnal patterns of temperature

The diurnal template used for generation of hourly temperatures from Tmin and Tmax is shown in Figure. 4.

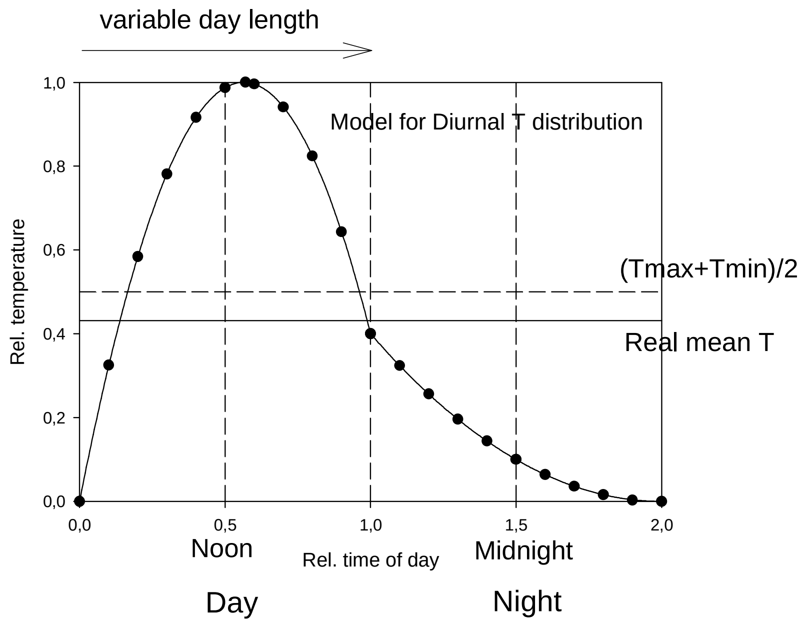


Figure. 4. Relative diurnal pattern (template) for temperature distribution used to calculate hourly temperatures from daily min and max temperatures of the entities concerned (air, water or panicle). Calculated day length is used to « stretch » or compress the relative day or night periods. Empirical equations used are [$Y=3.55\ X – 3.15\ X^2$] (day) and [$0.4 - 0.8\ X + 0.4\ X^2$] (night).

1.12 Simulation of LAI and LTR

LAI simulation is needed because the calculation of floodwater and panicle temperature requires information on ground cover. In the absence of biomass simulation in RIDEV, we chose to drive LAI with the following information:

- Thermal time expressed as fraction of potential daily thermal time, generated by the model [Tphys / (Topt – Tbase)], range 0…1
- Initial LAI as a function of population POP and individual initial plant leaf area PLAini: [LAI = POP * PLAini / 10000]. POP and PLAini are parameters to be set by users. POP range is typically 50000…500000 plants/ha and PLAini is in the order of 0.0001 m^2.
- Maximal leaf relative growth rate (LRGRmax), range 1.1…1.5 (resulting in daily multiplication of LAI by 1.1 to 1.5 during exponential growth). Thus, [LAI = LAI * LRGRmax * Tphys/(Topt-Tbase)]. LRGRmax is a crop parameter.
- Once LAI exceeds 1, linear growth is implemented using [LAI = LAI + (LRGRmax – 1) * 1 * Tphys / (Topt – Tbase)]
- LAI is capped by the parameter LAImax
- If the transplanting option is chosen by user, a transplanting shock can be implemented that delays both LAI and phenological development. The user chooses the duration of the shock (parameter DDtransplantingShock, in degree-days, range 0…100), during which development stops and leaf growth is halved.

This simple model can easily be calibrated to approximate common LAI patterns (post-flowering dynamic are of no importance to RIDEV). No particular accuracy is needed because LAI simulation only serves to estimate ground cover for water and panicle temperature calculation.

Light transmission ratio (LTR) is calculated with Lambert-Beer's law, using the value of 0.6 for the extinction coefficient:

LTR = exp (-0.6 * LAI)

LTR ranges from 0 (dense crop) to 1 (no crop).

1.13 Simulation of spikelet sterility

Thermal sterility is simulated on a range from 0 (none) to 1 (total). It is simulated for 3 different, successive sensitive phases, each having a specific sensitivity set by a crop parameter:

1. Microspore stage: From 280 to 140 dd before flowering (sensitivity to cold based on minimum water temperature; parameter CritSterCold1, range ca. 20 ^oC…10 ^oC)
2. Panicle exertion process: From 140 dd before flowering to flowering (sensitivity to cold based on minimum air temperature; parameter CritSterCold2, range ca. 20 ^oC…10 ^oC)
3. Flowering stage: From 50 dd before flowering until 50dd after flowering (sensitivity to heat based on temperature of the panicle at the hour of mid-anthesis; parameter CritSterHeat, range ca. 30 ^oC…40 ^oC)

The mean thermal conditions during the sensitive phases are used to calculate the sterility fractions. The 3 crop parameters set the temperature (^oC) at which the sterility effect sets on, based on a constant progression of 0.2 per ^oC (resulting in 100 % sterility as the temperature progresses by 5 ^oC beyond the critical value (parameter).

A 4th, constant source of sterility is simulated which stands for a genotypic baseline sterility (parameter SterBase, range 0…1). Typical values are between 0.1 and 0.2. A value of 1.0 would indicate genetic sterility as in male-sterile breeding lines.

The calculation of total sterility (base, SterCold1, SterCold2, SterHeat) is not additive because it cannot exceed the value 1. It is calculated as follows:

$$SterTot = SterBase + (1 - SterBase) * (1-[(1-SterCold1) * (1-SterCold2) * (1-SterHeat)])$$

1.14 Model parameters, input variables and output variables

1.14.1 Model parameters

There are 10 model parameters setting the general conditions of the simulation, 8 setting conditions for crop phenology (of which not all are always required, depending on choice of PP model), and 4 setting conditions for sterility.

<u>General conditions:</u>

Latitude	XX.XX, decimal
SowingDate	dd/mm/yyyy
Flooding	0 or 1 binary
Transplanting	0 or 1 binary
DDTransplantingShock	0…100 (dd)
POP	50000…500000 (plants per ha)
PLAini	ca. 0.0001 (m^2)
LRGRmax	1.1…1.5 (adjusted visually to obtain appropriate LAI dynamics)
LAImax	2…12
Z	Fraction of panicle height over canopy height (0.6…1)

<u>Phenology:</u>

BVPsum	duration of BVP (200…800 dd)	can be optimized
RPRsum	duration of RPR (200…500 dd0	never optimize together with BVP; set to 350
MATsum	duration of MAT (200…500 dd)	can be optimized

PPsens	)	
PPcrit	)	occur in different combinations in the 4 photoperiodism models
PPexp	)	to choose from. Value ranges as described further up. Can be optimized
SeuilPP	)	

<u>Sterility:</u>

SterBase	0…1, typically 0.1 to 0.2	)	
SterCold1	20…10 ^{o}C	)	Can be optimized (rice)
SterCold2	20…10 ^{o}C	)	
SterHeat	30…40 ^{o}C	)	

Only phenology and sterility parameters can be optimized and can be selected on the optimization menu (Annexe 2). The other parameters must then be fixed manually beforehand. Optimization can be performed as multi-fitting across different environments, cultural practices, population densities etc., but always for the same genotype (variety). A minimum diversity of sowing dates and/or latitudes (thus, day length and thermal conditions must be in the sample to allow meaningful optimizations of phenology.

Optimization scenarios are read from an external spreadsheet.

1.14.2 Input variables from external source file (formats in Annexe 3)

Tmin	daily minimum air T (^{o}C)
Tmax	daily maximum air T (^{o}C)
RHmin	daily minimum relative humidity (%)
Rs	daily cumulative solar global radiation (MJ)

(Optimizations also require observed crop data: duration from sowing to flowering (days), duration from sowing to maturity (days), and sterility (fraction).

1.14.3 Output variables

DSowEndBVP	Days from sowing to end of BVP
DSowPi	Days from sowing to PI
DSowFlowering	Days from sowing to flowering
DSowMaturity	Days from sowing to maturity
SterCold1Out	Cold sterility phase 1 (microspore)
SterCold2Out	Cold sterility phase 2 (panicle exertion process)
SterHeatOut	Heat sterility (flowering period)
SterTotOut	Total sterility

REFERENCES

Clerget, B., Dingkuhn, M., Chantereau, J., Hemberger, J., Louarn , G., Vaksmann, M., 2004. Does panicle initiation in tropical sorghum depend on day-to-day change in photoperiod? *Field Crops Res.* 88, 11-27. (Paper received ICRISAT Millenium Award in 2006)

Dingkuhn, M., A. Sow, A. Samb, S. Diack, F. Asch. 1995. Climatic determinants of irrigated rice performance in the Sahel. I. Photothermal and microclimatic responses of flowering. *Agricultural Systems* 48, 385-410.

Dingkuhn, M., K.M. Miezan. 1995. Climatic determinants of irrigated rice performance in the Sahel. II. Validation of photothermal constants and characterization of genotypes. *Agricultural Systems* 48, 411-434.

Dingkuhn, M. 1995. Climatic determinants of irrigated rice performance in the Sahel. III. Characterizing environments by simulating the crop's photothermal responses. *Agricultural Systems* 48, 435-456.

Dingkuhn, M. 1997. Characterizing irrigated rice environments using the rice phenology model RIDEV. In: K.M. Miezan, M.C.S. Wopereis, M. Dingkuhn, J. Deckers and T.F. Randolph (Eds.). Irrigated Rice in the Sahel: Prospects for Sustainable Development. West Africa Rice Development Association, B.P. 2551, Bouake 01, Cote d'Ivoire, ISBN 92 9113 1091. 343-360.

Dingkuhn, M., Kouressy, M., Vaksmann, M., Clerget, B., Chantereau, J. 2008. Applying to sorghum photoperiodism the concept of threshold-lowering during prolonged appetence. *European Journal of Agronomy* 28, 74-89.

Julia C, Dingkuhn M. 2012. Variation in time of day of anthesis in rice in different climatic environments. *European Journal of Agronomy* 43*, 166-174*.

Julia C., Dingkuhn M. 2012. Predicting heat induced sterility of rice spikelets requires simulation of crop-generated microclimate. *European Journal of Agronomy* (accepted).

Wopereis, MCS; Haefele, SM; Dingkuhn, M.; Sow, A. 2003. Decision support tools for irrgated rice-based systems in the Sahel. In Eds Struif Bontkes TE, Wopereis MCS. *A practical guide to decision-support tools for agricultural productivity and soil fertility enhancement in sub-Saharan Africa*. IFDC and CTA, Wageningen, The Netherlands, 114-126.

ANNEXE : Source code of RIDEV V2 (without interface and optimization utility)

```
void ridev(
            const double *alt, const double *latitude,
            const double *longitude, const int *days,
            const double *tmin, const double *tmax,
            const double *hrmin, const double *hrmax,
            const double *wind, const double *rg,
            const int *size, const double *POP,
            const double *PLAini, const int *Flooding,
            const int *Transplanting, const double *DDTransplantingShock,
            const double *SumBVP, const double *SumRPR,
            const double *SumMatu,  const double *lrgrmax,
            const double *TOpt, const double *TBase,
            const double *CritSterCold1,  const double *CritSterCold2,
            const double *CritSterHeat, const double *SterBase,
            const double *HrAnthBefNoon20C,  const double *PPExp,
            const double *PPCrit, const double *SeuilPP,
            const double *PPSens,  const int *PSPComputationMode,
            const double *Z, const double *LAIMax,
            double *DSowFlowering, double *DSowMaturity,
            double *SterHeatOut, double *SterCold1Out,
            double *SterCold2Out, double *SterTotOut,
            int *DSowEndBVP, int *DSowPi,
            double *AnthTimeMean, double *LAIs
            )
{


    const double RgMax = 22.;
    double SumBVPCor = 0.;
    double TMin, TWatMin, TPanMin = 0, TEntMin;
    double TMax, TWatMax, TPanMax = 0, TEntMax;
    double Rg;
    double TMoy;
    double TMoyPrec = 0.;
    double TPhys;
    double HrMin;
    double HrMax;
    double Wind;
    double ETP;
    int Day;
    double LatRad;
    double Decli;
    double SunPosi;
    double DayLength;
    double OldDayLength = 0.;
    int NumPhase = BVP;
    int DayCounter = 0;
    double SumTPhysRPR = 0;
    double SumTPhysBVP = 0;
    double SumTPhysPSP = 0;
    double SumTPhysMatu = 0;
    double SumTPhys = 0;
    int DaysToEndBVP = 0;
    int DaysToPI = 0;
    int DaysToFlowering = 0;
    int DaysToMaturity = 0;
    double TRefSterCold1 = 0;
    double TRefSterCold2 = 0;
    double SterCold1 = 0;
    double SterCold2 = 0;
    double SterHeat = 0;
    double SterTot = 0;
    double LAI = 0;
    double LTR = 0;
    double AnthesisTimeMean = 0;
    double MicroSterMeanCounter = 0;
    double PanSterMeanCounter = 0;
    double AnthTimeMeanFlag = 0;
    int i = 0;
    int HeatSterilityCounter = 0;
    int TransplantingShock;




    ComputeInitialLAI(POP, PLAini, LAI);

    ComputeBVPExtension(Transplanting, SumBVP, DDTransplantingShock,
                        SumBVPCor);


    while ((i < *size) and (NumPhase < End)) {

        TMin = tmin[i];
        TMax = tmax[i];
        Rg = rg[i];
        HrMin = hrmin[i];
        HrMax = hrmax[i];
        Wind = wind[i];
        Day = days[i];


        DayCounter = ComputeIncDayCounter(DayCounter);
        ComputeTMoy(TMin, TMax, TMoy);


        if (*Flooding != 0) {
            ComputeETo(alt, lrgrmax, Rg, TMin, TMax, HrMin, HrMax, TMoy,
                        TMoyPrec, Wind, ETP);
        }

        ComputeTMoyPrec(TMoy, TMoyPrec);
        ComputeDegToRad(latitude, LatRad);
        ComputeDecli(Day, Decli);
        ComputeSunPosi(LatRad, Decli, SunPosi);
        ComputeDayLength(SunPosi, DayLength);
        ComputeLTR(LAI, LTR);


        if (*Flooding != 0) {
            ComputeTWaterMinTWaterMax(TMin, TMax, LTR, TWatMin, TWatMax);

            ComputeTPanMinTPanMax(TMin, TMax, Rg, RgMax, Z, HrMin, LTR,
                                  DayLength, TPanMin, TPanMax);
        }

        ComputeReferenceEntityForTPhys(NumPhase, Flooding, TMin, TMax,
                                       TWatMin, TWatMax, SumTPhysRPR, SumRPR,
                                       TEntMin, TEntMax);

        ComputeTPhys(DayLength, TOpt, TBase, TEntMin, TEntMax, TPhys);

        ComputePhases(NumPhase, PSPComputationMode, DayCounter, SumTPhys,
                      SumTPhysBVP, SumBVPCor, DaysToEndBVP, TPhys,
                      SumTPhysPSP, OldDayLength, DayLength, DaysToPI,
                      SumTPhysRPR, SumRPR, DaysToFlowering, SumTPhysMatu,
                      SumMatu, DaysToMaturity,
                      PPExp, PPCrit, SeuilPP, PPSens);

        TransplantingShock = ComputeTransplantingShock(Transplanting,
                                                      NumPhase, SumTPhysBVP,
                                                      DDTransplantingShock);

        LAI = ComputeLAI(NumPhase, LAI, lrgrmax, TOpt, TBase, TPhys,
                         TransplantingShock, LAIMax);


        ComputeMicrosporeColdSterility(TEntMin, SumTPhysRPR, SumRPR,
                                       CritSterCold1, TRefSterCold1,
                                       SterCold1, MicroSterMeanCounter);

        ComputePanicleColdSterility(TEntMin, SumTPhysRPR, SumRPR,
                                    CritSterCold2, TRefSterCold2, SterCold2,
                                    PanSterMeanCounter);

        if (*Flooding != 0) {

                    ComputeHeatSterility(SumTPhysRPR, SumTPhysMatu, SumRPR,
                                         CritSterHeat, DayLength,
                                         HrAnthBefNoon20C, TMin, TPanMin,
                                         TPanMax, HeatSterilityCounter,
                                         SterHeat, AnthesisTimeMean,
                                         AnthTimeMeanFlag);

        }

        ComputeOldDayLength(DayLength, OldDayLength);

        LAIs[i] = LAI;
        i++;

    }


    if (HeatSterilityCounter != 0) {
```

```
            SterHeat = SterHeat / HeatSterilityCounter;

        }


        SterTot = ComputeTotalSterility(SterBase, SterCold1, SterCold2,
                                        SterHeat);

        *DSowFlowering = DaysToFlowering;
        *DSowMaturity = DaysToMaturity;

        if( *Flooding != 0 )
            *SterHeatOut = SterHeat;
        else
            *SterHeatOut = 999;


        *SterCold1Out = SterCold1;
        *SterCold2Out = SterCold2;
        *SterTotOut = SterTot;
        *DSowEndBVP = DaysToEndBVP;
        *DSowPi = DaysToPI;
        *AnthTimeMean = AnthesisTimeMean;


    }

    void ComputeInitialLAI(
                            const double *POP, const double *PLAini,
                            double &LAI
                            )
    {
        LAI = (*POP * *PLAini) / 10000;
    }

    void ComputeBVPExtension(
                            const int *Transplanting, const double *SumBVP,
                            const double *DDTransplantingShock,
                            double &SumBVPCor
                            )
    {
        if (*Transplanting == 1) {

            SumBVPCor = *SumBVP + *DDTransplantingShock;

        } else {

            SumBVPCor = *SumBVP;

        }
    }
```

```
int ComputeTransplantingShock(
                              const int *Transplanting, const int NumPhase,
                              const double SumTPhysBVP,
                              const double *DDTransplantingShock
                              )
{
    if (
        (*Transplanting == 1) and
        (NumPhase == 1) and
        (SumTPhysBVP > 250.) and
        (SumTPhysBVP < (SumTPhysBVP + *DDTransplantingShock))
      ){

        return 1;

    } else {

        return 0;

    }
}

void ComputeTMoy(
                  const double TMin, const double TMax,
                  double &TMoy)
{

    TMoy = (TMin + TMax) / 2.;

}

void ComputeETo(
                  const double *alt, const double *lrgrmax,
                  const double Rg, const double TMin,
                  const double TMax, const double HrMin,
                  const double HrMax, const double TMoy,
                  const double TMoyPrec, const double Wind,
                  double &ETP
                  )
{

    double eSat = 0.3054 * (std::exp(17.27 * TMax / (TMax + 237.3)) +
                  std::exp(17.27 * TMin / (TMin + 237.3)));

    double eActual = 0.3054 * (std::exp(17.27 * TMax / (TMax + 237.3)) *
                     HrMin / 100 + std::exp(17.27 * TMin / (TMin + 237.3)) *
                     HrMax / 100);

    double RgRgMax = std::min(Rg / *lrgrmax, 1.);

    double Rn = 0.77 * Rg - (1.35 * RgRgMax - 0.35) * (0.34 - 0.14 *
                std::pow(eActual, 0.5)) * (std::pow(TMax + 273.16, 4) +
                std::pow(TMin + 273.16, 4)) * 2.45015 * std::pow(10, -9);

    double TLat = 2.501 - 2.361 * std::pow(10, -3) * TMoy;
    double delta = 4098 * (0.6108 * std::exp(17.27 * TMoy / (TMoy + 237.3)))
                  / std::pow(TMoy + 237.3, 2);
    double Kpsy = 0.00163 * 101.3 * std::pow(1 - (0.0065 * *alt / 293), 5.26)
                  / TLat;

    double G = 0.38 * (TMoy - TMoyPrec);

    double Erad = 0.408 * (Rn - G) * delta / (delta + Kpsy * (1 + 0.34 *
                  Wind));

    double Eaero = (900 / (TMoy + 273.16)) * ((eSat - eActual) * Wind) * Kpsy
                  / (delta + Kpsy * (1 + 0.34 * Wind));

    ETP = Erad + Eaero;

}

void ComputeTMoyPrec(
                        const double TMoy, double &TMoyPrec
                        )
{

    TMoyPrec = TMoy;

}

void ComputeOldDayLength(
                        const double DayLength, double &OldDayLength
                        )
{

    OldDayLength = DayLength;

}

void ComputeDegToRad(
                        const double *latitude, double &LatRad
                        )
{

    LatRad = *latitude * M_PI / 180;

}

void ComputeDecli(
                        const double Day, double &Decli
                  )
{

    Decli = 0.409 * std::sin(0.0172 * Day - 1.39);

}
```

```
void ComputeSunPosi(
                        const double LatRad, const double Decli,
                        double &SunPosi
                        )
{

    SunPosi = std::acos(-std::tan(LatRad) * std::tan(Decli));

}

void ComputeDayLength(
                        const double SunPosi, double &DayLength
                        ) {

    DayLength = 7.64 * SunPosi;

}

void ComputeTWaterMinTWaterMax(
                              const double TMin, const double TMax,
                              const double LTR, double &TWatMin,
                              double &TWatMax
                              )
{

    TWatMin = TMin + 0.658 * (TMax - TMin) * (1 - LTR) + 0.425 * (TMax –
              TMin) * LTR - 0.303 * std::pow((TMax - TMin), 1.2);

    TWatMax = TMax + (-0.728 * (TMax - TMin) * (1 - LTR) + 0.275 * (TMax –
              TMin) * LTR) * (TMax - 18.46) / 20;


}

void ComputeTPanMinTPanMax(
                              const double TMin, const double TMax,
                              const double Rg, const double RgMax,
                              const double *Z, const double HrMin,
                              const double LTR, const double DayLength,
                              double &TPanMin, double &TPanMax
                              )
{

    TPanMin = TMin;

    double WattPerSqm = 1000. * (Rg * 12. / DayLength) / RgMax;

    TPanMax = TMax - (0.78 + 0.422 * TMax - 0.0443 * HrMin -0.00287 *
              WattPerSqm - 8.05 * *Z -6.59 * LTR);

}
```

```
void ComputeReferenceEntityForTPhys(
                                    const int NumPhase, const int *Flooding,
                                    const double TMin, const double TMax,
                                    const double TWatMin,
                                    const double TWatMax,
                                    const double SumTPhysRPR,
                                    const double *SumRPR, double &TEntMin,
                                    double &TEntMax
                                    )
{

    if (*Flooding == 0) {

        TEntMin = TMin;
        TEntMax = TMax;

    } else if (NumPhase < RPR) {

        TEntMin = TWatMin;
        TEntMax = TWatMax;

    } else if (NumPhase == RPR) {

        TEntMin = ((1 - SumTPhysRPR / *SumRPR) * TWatMin + (SumTPhysRPR /
                  *SumRPR) * TMin) ;

        TEntMax = ((1 - SumTPhysRPR / *SumRPR) * TWatMax + (SumTPhysRPR /
                  *SumRPR) * TMax) ;

    } else {

        TEntMin = TMin;
        TEntMax = TMax;

    }

}
```

```
double ComputeHourlyTEnt(
                        const double DayLength, const double Time,
                        const double TEntMin, const double TEntMax
                        )
{

    double TimeSR = (24 - DayLength) / 2.;
    double HAfterSR = Time - TimeSR;
    double TRel;

    if (HAfterSR <= 0) {

        HAfterSR = HAfterSR + 24;

    }

    if (HAfterSR <= DayLength) {

        double TimeFracDay = HAfterSR / DayLength;

        TRel = 3.55 * TimeFracDay - 3.15 * std::pow(TimeFracDay, 2);

    }
    else {

        double TimeFracNight = (HAfterSR - DayLength) / (24 - DayLength);

        TRel = 0.4 - 0.8 * TimeFracNight + 0.4 * std::pow(TimeFracNight, 2);

    }

    return (TEntMin + TRel * (TEntMax - TEntMin));

}


void ComputeTPhys(
                  const double DayLength, const double *TOpt,
                  const double *TBase, const double TEntMin,
                  const double TEntMax, double &TPhys
                  )
{

    TPhys = 0;

    for (int i = 0; i < 24; i++) {

        double TEnt = ComputeHourlyTEnt(DayLength, i + 1, TEntMin, TEntMax);
        double TPhysH = std::max(std::min(TEnt, *TOpt) - *TBase, 0.);

        TPhys += TPhysH;

    }

    TPhys = TPhys / 24.;

}
```

```
int ComputeIncDayCounter(const int DayCounter) {

    return DayCounter + 1;

}

void ComputePhases(
                        int &NumPhase, const int *PSPComputationMode,
                        const int Counter, double &SumTPhys,
                        double &SumTPhysBVP, const double SumBVPCor,
                        int &DaysToEndBVP, const double TPhys,
                        double &SumTPhysPSP, const double OldDayLength,
                        const double DayLength, int &DaysToPI,
                        double &SumTPhysRPR, const double *SumRPR,
                        int &DaysToFlowering, double &SumTPhysMatu,
                        const double *SumMatu, int &DaysToMaturity,
                        const double *PPExp, const double *PPCrit,
                        const double *SeuilPP, const double *PPSens
                  )
{

    if (NumPhase < PSP) {

        SumTPhys += TPhys;

        SumTPhysBVP += TPhys;

        if (SumTPhysBVP >= SumBVPCor) {

            NumPhase = PSP;

            DaysToEndBVP = Counter;

        }

    } else if (NumPhase == PSP) {

        double dPSP;
        double dDayLength;

        switch (*PSPComputationMode) {

            case 1:

                dPSP = *PPSens * TPhys / std::max(DayLength - *PPCrit,
                        0.001);

                SumTPhys += TPhys;
                SumTPhysPSP += dPSP;
                if (SumTPhysPSP >= 1.) {

                    NumPhase = RPR;
                    DaysToPI = Counter;

                }
                break;
            case 2:
                dPSP = *PPSens * TPhys / std::pow(std::max(DayLength –
                       *PPCrit, 0.001), *PPExp);

                SumTPhys += TPhys;
                SumTPhysPSP += dPSP;

                if (SumTPhysPSP >= 1.) {

                    NumPhase = RPR;
                    DaysToPI = Counter;

                }
                break;

            case 3:
                dDayLength = DayLength - OldDayLength;

                dPSP = std::max(*PPSens * TPhys * (*PPCrit - dDayLength),
                       0.);

                SumTPhys += TPhys;
                SumTPhysPSP += dPSP;

                if (SumTPhysPSP >= 1.) {

                    NumPhase = RPR;
                    DaysToPI = Counter;

                }
                break;

            case 4:
                double SumPP = std::pow((1000. / SumTPhysPSP), *PPExp) *
                               std::max(0., DayLength - *PPCrit) / (*SeuilPP
                               - *PPCrit);

                SumTPhys += TPhys;
                SumTPhysPSP += TPhys;

                if (SumPP <= *PPSens) {

                    NumPhase = RPR;
                    DaysToPI = Counter;

                }
                break;

        }

    } else if (NumPhase == RPR) {

        SumTPhys +=  TPhys;
        SumTPhysRPR += TPhys;
```

```
        if (SumTPhysRPR >= *SumRPR) {

            NumPhase = Matu;
            DaysToFlowering = Counter;

        }
    } else if (NumPhase == Matu) {

        SumTPhys += TPhys;
        SumTPhysMatu += TPhys;

        if (SumTPhysMatu >= *SumMatu) {

            NumPhase = End;
            DaysToMaturity = Counter;

        }

    }

}

double ComputeLAI(
                  const int NumPhase, const double LAI,
                  const double *LRGRMax, const double *TOpt,
                  const double *TBase,
                  const int TPhys, const int TransplantingShock,
                  const double *LAIMax
                  )
{

    double newLAI;

    if ((LAI <= 1) and (TransplantingShock == 0) and (NumPhase < 4)) {

        newLAI = LAI * (1 + (*LRGRMax - 1) * TPhys / (*TOpt - *TBase));

    } else if ((LAI <= 1) and (TransplantingShock == 1) and (NumPhase < 4)) {

        newLAI = LAI * (1 + (*LRGRMax - 1) * 0.5 * TPhys / (*TOpt - *TBase));

    } else if ((LAI > 1) and (LAI <= *LAIMax)) {

        newLAI = LAI + (*LRGRMax - 1) * 1 * TPhys / (*TOpt - *TBase);

    } else {

        newLAI = LAI;

    }

    return std::min(newLAI, *LAIMax);

}
```

```
void ComputeLTR(
                  const double LAI, double &LTR
                  )
{

    if (LAI == 0) {

        LTR = 1.;

    } else {

        LTR = std::exp(-0.6 * LAI);

    }
}


void ComputeMicrosporeColdSterility(
                                    const double TEntMin,
                                    const double SumTPhysRPR,
                                    const double *SumRPR,
                                    const double *CritSterCold1,
                                    double &TRefSterCold1,
                                    double &SterCold1,
                                    double &MicroSterMeanCounter
                                    )
{

    if (
            (SumTPhysRPR >= (*SumRPR * 0.3)) and
            (SumTPhysRPR <= (*SumRPR * 0.7))
       )
    {

        TRefSterCold1 = TEntMin;

        SterCold1 += std::min(1., std::max(0., (*CritSterCold1 -
                     TRefSterCold1) / 5.));

        MicroSterMeanCounter++;

    } else if(
                  (MicroSterMeanCounter != -1) and
                  (SumTPhysRPR > (*SumRPR * 0.7))
              )
    {
        SterCold1 /= MicroSterMeanCounter;

        MicroSterMeanCounter = -1;
    }

}
```

```
void ComputePanicleColdSterility(
                                    const double TEntMin,
                                    const double SumTPhysRPR,
                                    const double *SumRPR,
                                    const double *CritSterCold2,
                                    double &TRefSterCold2,
                                    double &SterCold2,
                                    double &PanSterMeanCounter
                                    )
{

    if (
            (SumTPhysRPR > (*SumRPR * 0.7)) and
            (SumTPhysRPR <= *SumRPR)
        )
    {

        TRefSterCold2 = TEntMin;

        SterCold2 += std::min(1., std::max(0., (*CritSterCold2 -
                     TRefSterCold2) / 5.));

        PanSterMeanCounter++;

    } else if(
                  (PanSterMeanCounter != -1) and
                  (SumTPhysRPR > *SumRPR )
                  )
    {

        SterCold2 /= PanSterMeanCounter;
        PanSterMeanCounter = -1;

    }

}
```

```
void ComputeHeatSterility(
                              const double SumTPhysRPR,
                              const double SumTPhysMatu,
                              const double *SumRPR,
                              const double *CritSterHeat,
                              const double DayLength,
                              const double *HrAnthBefNoon20C,
                              const double TMin, const double TPanMin,
                              double const TPanMax,
                              int &HeatSterilityCounter, double &SterHeat,
                              double &AnthesisTimeMean ,
                              double &AnthTimeMeanFlag
                        )
{

    if (
            (SumTPhysRPR >= (*SumRPR - 60)) and
            (SumTPhysMatu <= 60)
            )
   {
        double AnthesisTime = (14. - 0.4 * TMin) + (24 - DayLength) / 2 -
                              *HrAnthBefNoon20C;

        AnthesisTimeMean += AnthesisTime;

        double TPanAnth = ComputeHourlyTEnt(DayLength, AnthesisTime, TPanMin,
                          TPanMax);

        double SterHeatDay = std::min(1., std::max(0., (TPanAnth -
                             *CritSterHeat) / 5.));

        SterHeat += SterHeatDay;

        ++HeatSterilityCounter;

        if(AnthTimeMeanFlag == 0) AnthTimeMeanFlag = 1;

    } else {

        if( AnthTimeMeanFlag == 1 )
        {

            AnthTimeMeanFlag = -1;
            AnthesisTimeMean /= HeatSterilityCounter;

        }
    }
}
```

```
double ComputeTotalSterility(
                              const double *SterBase, const double SterCold1,
                              const double SterCold2, const double SterHeat
                              )
{

    return (*SterBase + (1. - *SterBase) * (1. - ((1. - SterCold1) * (1. -
            SterCold2) * (1. - SterHeat))));

}
```