Implementing a quasi-analytical method for accelerating the spin-up into ORCHIDEE

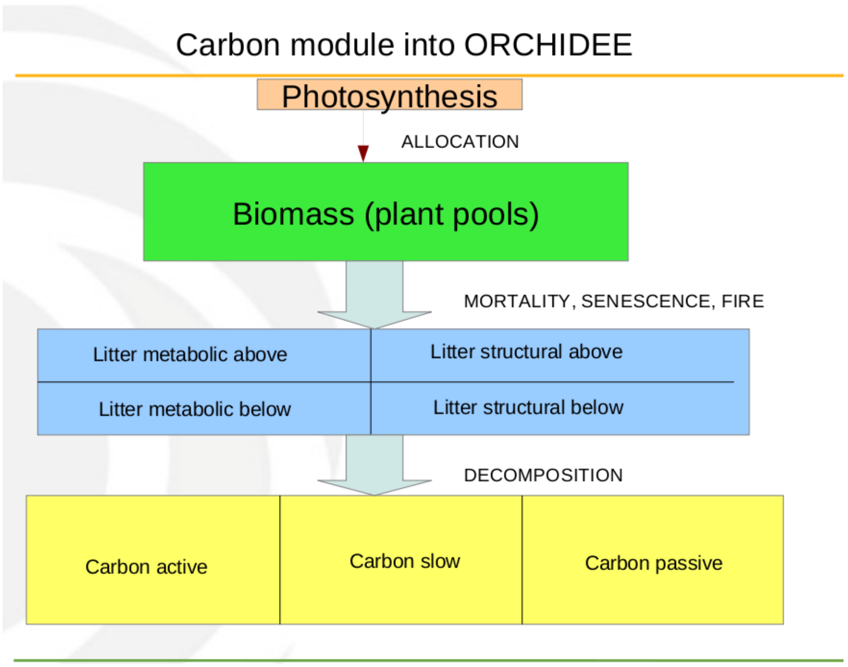
Didier Solyga

1 Introduction : the spin-up problem

Before performing a simulation with ORCHIDEE, we should create an initial state for the studied ecosystem(s). We assume that the initial state results from carbon accumulation over the past centuries. So the common approach consists to set to the different carbon pools at steady state by running the full model iteratively over a given forcing period (hundred to thousand years). This stage is called spin-up. By its definition the spin-up process is computationally expensive. In order to solve this problem, some methods have been developped. This document presents a new method to solve the spin-up problem for ORCHIDEE, based on algebra and matrix sequences.

2 Spin-Up in the current ORCHIDEE version (AR5/1.9.6)

2.1 Organization of the carbon module into ORCHIDEE



The different carbon pools have various residence time. The slowest pool is the passive pool.

2.2 Native methods

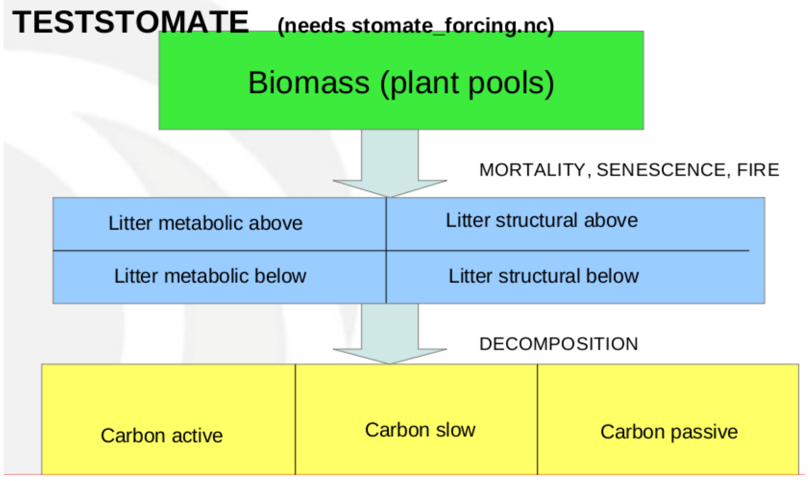
The native method consists to run ORCHIDEE a very long time (thousands of years) by using the same forcing period. For the SPINUP configuration, it consists to use the OOL_SEC_STO configuration only. This method is both reliable and computationnally expensive due to the residence time of the passive pool, which is the limiting factor for this method.

2.3 Other methods : decoupling the carbon pools

A new approach was implemented to reduce the cost of the traditional method. The strategy adopted consists to divide the spin-up into two sub-systems : one to accelerate the convergence of the biomass pools, the other one to accelerate the convergence of the soil pools. Two specific subprograms had been developped : TESTSTOMATE and FORCESOIL. They are both detailed in the next sections.

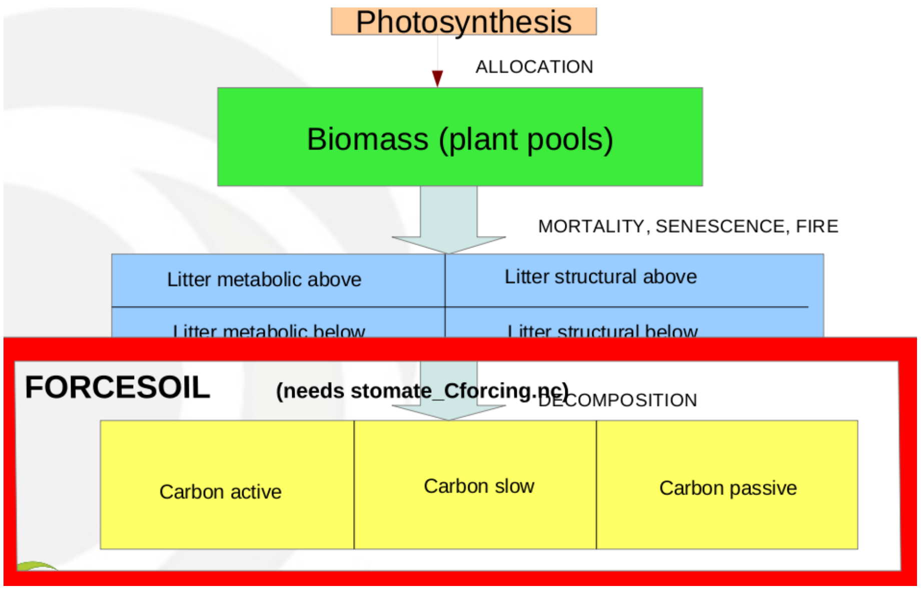
2.3.1 Teststomate program

TESTSTOMATE consists to set the biomass(plant pools) at steady state by activating STOMATE only. For this purpose, the program needs an forcing file(stomate_forcing.nc) created previously by a "full" ORCHIDEE run. TESTSTOMATE can be summarized by the following figure :



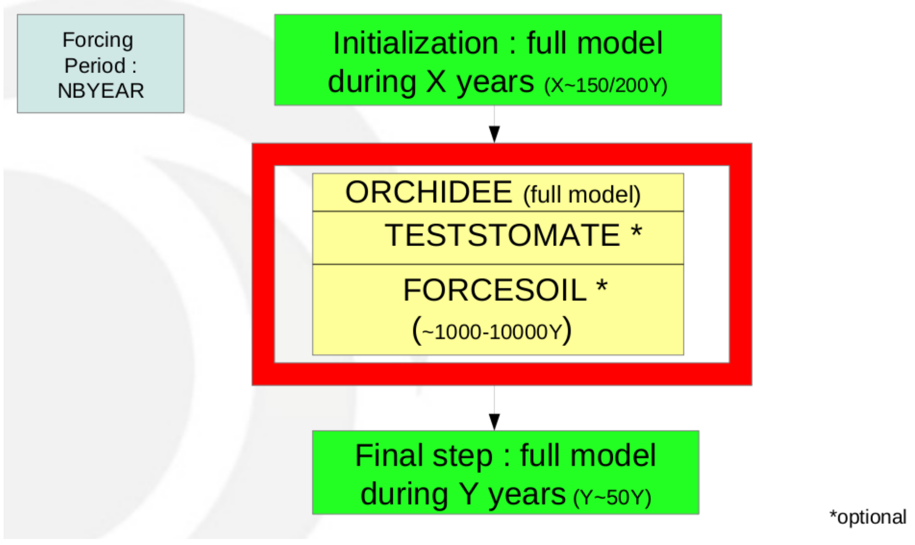
2.3.2 Forcesoil program

FORCESOIL sets the soil carbon pools at steady state by running iteratively stomate_soilcarbon subroutine. The number of iterations is ranged between 1000 and 10000 times (according to the setups). This program needs also a forcing file called stomate_Cforcing.nc created previously by ORCHIDEE or TESTSTOMATE. By its definition, FORCESOIL is not computationally expensive. The following figure represents the sub-system FORCESOIL :



2.3.3 Summary : performing a spin-up run

Generally, a spin-up run consists in three stages : ORCHIDEE + (ORCHIDEE/TESTSTOMATE and/or FORCESOIL)*n_iter + ORCHIDEE


Forcing Period : NBYEAR
Initialization : full model during X years (X~150/200Y)
ORCHIDEE (full model)
TESTSTOMATE *
FORCESOIL *
(~1000-10000Y)
Final step : full model during Y years (Y~50Y)
*optional

2.4 Problems known

TESTSTOMATE can lead to some instabilities because it skips the interactions between vegetation and physical processes. With FORCESOIL, the sub-system reached an equilibrium which is different from the one calculated by the full model. So the full model has to be runned after to stabilize the system. Moreover the specific forcing files (stomate_forcing.nc and stomate_Cforcing.nc) were poorly managed. There does not exist a common protocol to evaluate the equilibrium (differences between set-ups, no tool,...). So we need a more efficient approach for the spin-up problem combining reliability and reasonable computational time .

3 The spin-up method used by PASIM model

3.1 Aims

The method presented below was first developped by Romain Lardy for the PASIM model. It helps to accelerate the convergence of the soil carbon pools. For ORCHIDEE, this method is a substitute for FORCESOIL only. This method is quasi-analytical and is completely integrated into ORCHIDEE (no specific program needed).

3.2 Implementation in PASIM

PASIM is a model of prairial management developped by the INRA institute. The soil module of PASIM is based on the CENTURY model like ORCHIDEE. There are 5 carbon pools into PASIM : litter structural, litter metabolic, carbon active, carbon slow and carbon passive.

To solve the spin-up problem for PASIM, a new method has been developped by Lardy and al (2011). This method consists to solve matrix systems a certain number of times until the equilibrium is reached.

The exchanges between the carbon pools are represented by a matrix of size (5,5) for each grid cell. At the end of each forcing period, a linear system is solved by an algebraic method called Gauss-Jordan (direct inversion of a linear matrix system). That is why we will refer it at the algebraic or analytical method.

The method is stopped when a criterion is reached (relative error on the L2 norm of the current total carbon stock calculated and the previous one). PASIM takes into account the nitrogen cycle and a similar method is used for calculating the nitrogen stock.

Unlike ORCHIDEE, PASIM does not divide the spin-up problem into sub-systems. The matrix method can accelerate the convergence of all the carbon pools, but especially the passive one. We can consider this one as an accelerated native method. See section 5 to understand how it was done for ORCHIDEE.

3.3 Limits

Contrary to ORCHIDEE, PASIM considers C3 grass only(PFT 10 in ORCHIDEE) and works on site scale. We know that the spin-up length differs from one PFT to another. Inside ORCHIDEE, the two litter pools are subdivided into two compartments : above and below. So ORCHIDEE in its standard version considers 7 pools : litter structural above and below, litter metabolic above and below, carbon active, carbon slow and carbon passive. Moreover, we think using L2 norm (vector norm) of the total stock to calculate the relative error not relevant. So it was decided to use the absolute value on the passive carbon pool.

4 Mathematical Principles

In the following section, we denote by C the vector of carbon pools used by the model. Notice that the method does not depend of the number of carbon pools considered because PASIM uses 5 pools and ORCHIDEE 7 pools. $C(t)$ follows the differential equation :

$$C'(t) = \rho(t) \cdot A(t) \cdot C(t) + B(t) \tag{1}$$

This equation cannot be solved directly because the function ρ is not continuous into ORCHIDEE.

We can approximate $C'(t)$ by the following formula :

$$C'(t) = \frac{C(t+dt) - C(t)}{dt} \tag{2}$$

where :

1. $\rho(t)$represents the water stress and temperature effects, that are assumed independent (unitless). ($\rho(t)$ is the product of the water stress by the temperature effects).
2. $A(t)$ is the matrix of decomposition rates for soil organic matter at each time step (day^{-1}).
3. $B(t)$ is the vector of C litter input ($gC \cdot m^{-2} \cdot day^{-1}$).

If $t = n \cdot dt$, we set :

$$C(t) = C(n \cdot dt) = C_n \tag{3}$$

$$B(t) = B(n \cdot dt) = B_n \tag{4}$$

$$\rho(t) = \rho(n \cdot dt) = \rho_n \tag{5}$$

$$A(t) = A(n \cdot dt) = A_n \tag{6}$$

So we have :

$$C_{n+1} = (I + \rho_n \cdot dt \cdot A_n) \cdot C_n + dt \cdot B_n \tag{7}$$

We set :

$$D_n = (I + \rho_n \cdot dt \cdot A_n)$$

and :

$$E_n = B_n \cdot dt$$

We got :

$$C_{n+1} = D_n \cdot C_n + E_n$$

By the principle of mathematical induction, we obtain (proof in annex) :

$$C_n = \sum_{j=0}^{n-1} (\prod_{k=j+1}^{n-1} D_k) E_j + (\prod_{j=0}^{n-1} D_j) C_0$$

Let us define now the two following sequences :

$$V_0 = D_0 = I + \rho_0 \cdot dt \cdot A_0$$

$$V_n = D_n V_{n-1} = (I + \rho_n \cdot dt \cdot A_n) \cdot V_{n-1} = \prod_{k=0}^{n} (I + \rho_k \cdot dt \cdot A_k)$$

$$U_0 = E_0 = dt \cdot B_0$$

$$U_n = D_n U_{n-1} + E_n = (I + \rho_n \cdot dt \cdot A_n) U_{n-1} + B_n \cdot dt$$

So,

$$C_n = U_n + V_n C_0$$

Equilibrium means that C_n is a fixed point of the previous equation. When $C_n = C_0 = C^*$, we have to solve the following linear equation :

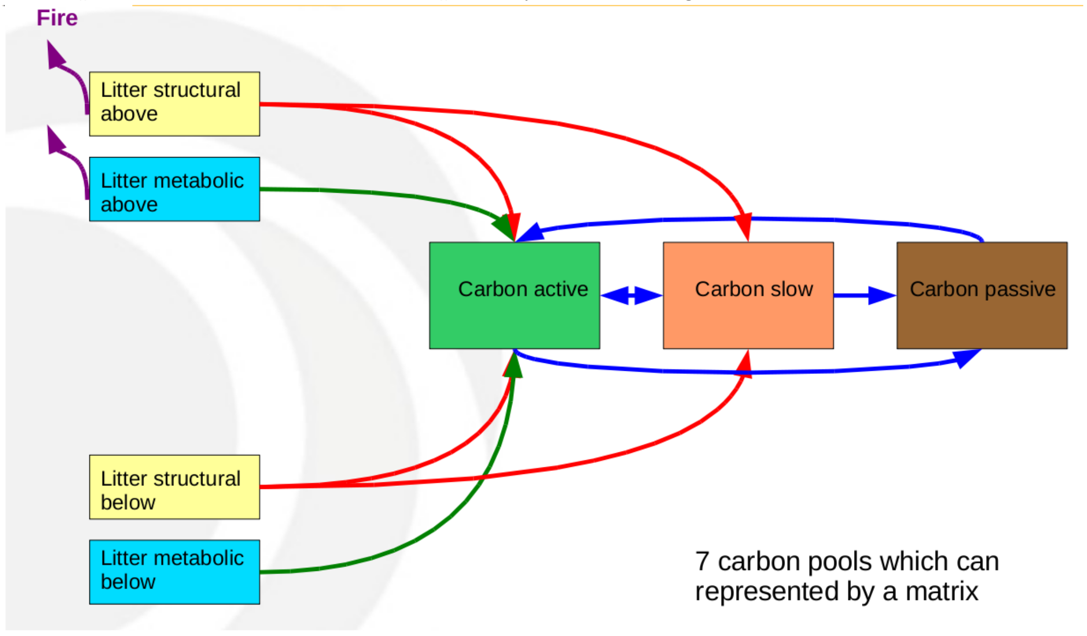
$$(I - V_n)C^* = U_n \tag{8}$$

As the matrix $I - V_n$ has no specific properties, we use the Gauss-Jordan algorithm to solve the previous equation. The Gauss-Jordan method has a complexity of $O(n^3)$ where n is the size of the matrix, which is reasonable. For information, the less expensive numerical method to solve a linear system has a complexity of $O(\frac{2}{3}n^3)$. Some methods are less complex but in this case, the matrix has a specific structure (ie symetric, etc...).

5 Implementation in ORCHIDEE

5.1 Modeling

As it was written in the previous section, ORCHIDEE considers 7 carbon pools. The components of the unknown vector C are (by order) : litter structural above, litter structural below, litter metabolic above, litter metabolic below, carbon active, carbon slow and carbon passive. We can represent the fluxes between the pools by the following scheme :



The implementation of this method affects only the following modules :stomate, stomate_io(for restart variables), stomate_litter, stomate_soilcarbon, stomate_lpj and lpj_fire. One module called gauss_jordan_method was added in src_global. A global flag called SPINUP_ANALYTIC controls the activation of the method. This flag is defined in the module constantes.

As it was written above, ORCHIDEE has more PFTs than PASIM. So we have to solve a linear system for each pixel and for each PFT. We can skip the bare soil. So in its standard configuration, $npts * (nvm - 1)$ linear systems are needed to be solved where $npts$ denote the number of continental grid cells. As we know the complexity of the Gauss Jordan method, we can estimate the total number of operations in standard mode :

$$12 \times 7^3 \times npts = 4116 \times npts$$

where *npts* ranges between 15000 and 60000 according the forcing files.

NB : The method is independent of the number of the carbon pools.

5.2 Structure of the matrix

Organization of the matrix of fluxes exchanges :

$$\begin{array}{c|ccccccc|}
 & lit\_str\_ab & lit\_str\_be & lit\ met\_ab & lit\_met\_be & active & slow & passive \\
lit\_str\_ab & - & & & & & & \\
lit\_str\_be & & - & & & & & \\
lit\_met\_ab & & & - & & & & \\
lit\_met\_be & & & & - & & & \\
active & + & + & + & + & - & + & + \\
slow & + & + & & & + & - & \\
passive & & & & & + & + & -
\end{array}$$

Each line of the matrix matches to a litter or a carbon pool. Each of its pools receive fluxes coming from the others pools and emit fluxes. When a pool receive a flux, it is marked as positive contribution to the pool. If the pool emit a flux, it is marked as negative contribution of the pool. The blanks correponds to zero (no contribution).

Finally, the matrix $\rho(t) \cdot A(t)$ has the following structure :

$$\begin{pmatrix}
a_{11} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & a_{22} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & a_{33} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & a_{44} & 0 & 0 & 0 \\
a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\
a_{61} & a_{62} & 0 & 0 & a_{65} & a_{66} & 0 \\
0 & 0 & 0 & 0 & a_{75} & 0 & a_{77}
\end{pmatrix}$$

As we have explained $a_{ij} < 0$ when $i = j$ and > 0 otherwise.

The values of a_{ij} are described below.

Structure of vectorB :

$$\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

b_1, b_2, b_3, b_4 correspond to litter input.

The unknow vector C used in the code has the following components :

$$\begin{pmatrix} litter\_structural\_above \\ litter\_structural\_below \\ litter\_metabolic\_above \\ litter\_metabolic\_below \\ carbon\_active \\ carbon\_slow \\ carbon\_passive \end{pmatrix}$$

In the code, $\rho(t)$ is the product of the variables control_temp by control_moist.

5.3 modules modified

The following modules have been modified in the code : constantes, stomate, stomate_lpj, stomate_litter, stomate_soilcarbon, lpj_fire and stomate_io. forcesoil A new file has been added : gauss_jordan_method.f90.

The three modules : stomate_litter, stomate_soilcarbon and lpj_fire have a similar goal which is to fill matrixA and vectorB variables.

5.3.1 constantes

Declaration of spinup_analytic. This flag controls the activation of the algebraic method.

```
LOGICAL, SAVE :: spinup_analytic = .TRUE.
!Config Key   = SPINUP_ANALYTIC
!Config Desc  = Activation of the analytic resolution of the spinup.
!Config If    = OK_STOMATE
!Config Def   = y
!Config Help  = Activate this option if you want to solve the spinup by the Gauss-Jordan method.
!Config Units = BOOLEAN
CALL getin_p('SPINUP_ANALYTIC',spinup_analytic)
```

NB : spinup_analytic could be declared in stomate module.

5.3.2 stomate_litter

```
IF(spinup_analytic) THEN

   MatrixA(:,:,:,:) = zero
   VectorB(:,:,:) = zero


   DO m = 2,nvm

      !- MatrixA : carbon fluxes leaving the litter

      MatrixA(:,m,istructural_above,istructural_above)= - dt/litter_tau(istructural) * &
           control_temp(:,iabove) * control_moist(:,iabove) * &
           exp( -litter_struct_coef * lignin_struc(:,m,iabove) )

      MatrixA(:,m,istructural_below,istructural_below) = - dt/litter_tau(istructural) * &
           control_temp(:,ibelow) * control_moist(:,ibelow) * &
           exp( -litter_struct_coef * lignin_struc(:,m,ibelow) )

      MatrixA(:,m,imetabolic_above,imetabolic_above) = - dt/litter_tau(imetabolic) * &
           control_temp(:,iabove) * control_moist(:,iabove)

      MatrixA(:,m,imetabolic_below,imetabolic_below) = - dt/litter_tau(imetabolic) * &
           control_temp(:,ibelow) * control_moist(:,ibelow)


      !- MatrixA : carbon fluxes between the litter and the pools

      MatrixA(:,m,iactive_pool,istructural_above) = frac_soil(istructural,iactive,iabove) * &
           dt/litter_tau(istructural) * &
           control_temp(:,iabove) * control_moist(:,iabove) * &
           exp( -litter_struct_coef * lignin_struc(:,m,iabove) ) * &
           ( 1. - lignin_struc(:,m,iabove) )


      MatrixA(:,m,iactive_pool,istructural_below) = frac_soil(istructural,iactive,ibelow) * &
           dt/litter_tau(istructural) * &
           control_temp(:,ibelow) * control_moist(:,ibelow) * &
           exp( -litter_struct_coef * lignin_struc(:,m,ibelow) ) * &
           ( 1. - lignin_struc(:,m,ibelow) )


      MatrixA(:,m,iactive_pool,imetabolic_above) =  frac_soil(imetabolic,iactive,iabove) * &
           dt/litter_tau(imetabolic) * control_temp(:,iabove) * control_moist(:,iabove)
```

```
            MatrixA(:,m,iactive_pool,imetabolic_below) =  frac_soil(imetabolic,iactive,ibelow) * &
                 dt/litter_tau(imetabolic) * control_temp(:,ibelow) * control_moist(:,ibelow)

            MatrixA(:,m,islow_pool,istructural_above) = frac_soil(istructural,islow,iabove) * &
                 dt/litter_tau(istructural) * &
                 control_temp(:,iabove) * control_moist(:,iabove) * &
                 exp( -litter_struct_coef * lignin_struc(:,m,iabove) )* &
                 lignin_struc(:,m,iabove)


            MatrixA(:,m,islow_pool,istructural_below) = frac_soil(istructural,islow,ibelow) * &
                 dt/litter_tau(istructural) * &
                 control_temp(:,ibelow) * control_moist(:,ibelow) *  &
                 exp( -litter_struct_coef * lignin_struc(:,m,ibelow) )* &
                 lignin_struc(:,m,ibelow)


            !- VectorB : carbon input -

            VectorB(:,m,istructural_above) = litter_inc_PFT(:,m,istructural,iabove)
            VectorB(:,m,istructural_below) = litter_inc_PFT(:,m,istructural,ibelow)
            VectorB(:,m,imetabolic_above) = litter_inc_PFT(:,m,imetabolic,iabove)
            VectorB(:,m,imetabolic_below) = litter_inc_PFT(:,m,imetabolic,ibelow)

            IF (bavard.GE.4) WRITE(numout,*) 'We␣filled␣MatrixA␣and␣VectorB'

         ENDDO ! Loop over # PFTs

      ENDIF ! spinup analytic
```

After this step, the following part of matrix A has been set :

$$\begin{pmatrix} a_{11} & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & 0 \\ 0 & 0 & 0 & a_{44} \\ a_{51} & a_{52} & a_{53} & a_{54} \\ a_{61} & a_{62} & 0 & 0 \end{pmatrix}$$

The vector B is completely set.

5.3.3 stomate_soilcarbon

```
      IF(spinup_analytic) THEN

         DO m = 2,nvm

            ! flux leaving the active pool
            MatrixA(:,m,iactive_pool,iactive_pool) = moins_un * &
                 dt/carbon_tau(iactive) * &
                 control_moist(:,ibelow) * control_temp(:,ibelow) * &
                 ( 1. - flux_tot_coeff(3) * clay(:))

            MatrixA(:,m,iactive_pool,islow_pool) =  frac_carb(:,islow,iactive)*dt/carbon_tau(islow) * &
                 control_moist(:,ibelow) * control_temp(:,ibelow)

            MatrixA(:,m,iactive_pool,ipassive_pool) =  frac_carb(:,ipassive,iactive) * &
                 dt/carbon_tau(ipassive) * control_moist(:,ibelow) * control_temp(:,ibelow)

            MatrixA(:,m,islow_pool,iactive_pool) =  frac_carb(:,iactive,islow) *&
                 dt/carbon_tau(iactive) * &
                 control_moist(:,ibelow) * control_temp(:,ibelow) * &
                 ( 1. - flux_tot_coeff(3) * clay(:) )

            ! flux leaving the slow pool
            MatrixA(:,m,islow_pool,islow_pool) = moins_un * &
                 dt/carbon_tau(islow) * &
                 control_moist(:,ibelow) * control_temp(:,ibelow)
```

```
MatrixA(:,m,ipassive_pool,iactive_pool) =  frac_carb(:,iactive,ipassive)* &
     dt/carbon_tau(iactive) * &
     control_moist(:,ibelow) * control_temp(:,ibelow) *&
     ( 1. - flux_tot_coeff(3) * clay(:) )

MatrixA(:,m,ipassive_pool,islow_pool) =  frac_carb(:,islow,ipassive) * &
     dt/carbon_tau(islow) * &
     control_moist(:,ibelow) * control_temp(:,ibelow)

! flux leaving the passive pool
MatrixA(:,m,ipassive_pool,ipassive_pool) =  moins_un * &
     dt/carbon_tau(ipassive) * &
     control_moist(:,ibelow) * control_temp(:,ibelow)


IF ( (.NOT. natural(m)) .AND. (.NOT. is_c4(m)) ) THEN ! C3crop


   ! flux leaving the active pool
   MatrixA(:,m,iactive_pool,iactive_pool) = MatrixA(:,m,iactive_pool,iactive_pool) * &
        flux_tot_coeff(1)

   MatrixA(:,m,iactive_pool,islow_pool)= MatrixA(:,m,iactive_pool,islow_pool) * &
        flux_tot_coeff(1)

   MatrixA(:,m,iactive_pool,ipassive_pool) = MatrixA(:,m,iactive_pool,ipassive_pool) * &
        flux_tot_coeff(1)


   MatrixA(:,m,islow_pool,iactive_pool) =  MatrixA(:,m,islow_pool,iactive_pool) * &
        flux_tot_coeff(1)


   ! flux leaving the slow pool

   MatrixA(:,m,islow_pool,islow_pool) = MatrixA(:,m,islow_pool,islow_pool) * &
        flux_tot_coeff(1)

   MatrixA(:,m,ipassive_pool,iactive_pool) = MatrixA(:,m,ipassive_pool,iactive_pool) * &
        flux_tot_coeff(1)

   MatrixA(:,m,ipassive_pool,islow_pool) = MatrixA(:,m,ipassive_pool,islow_pool) * &
        flux_tot_coeff(1)


   ! flux leaving the passive pool
   MatrixA(:,m,ipassive_pool,ipassive_pool) =  MatrixA(:,m,ipassive_pool,ipassive_pool) * &
        flux_tot_coeff(1)

ENDIF

IF ( (.NOT. natural(m)) .AND. is_c4(m) ) THEN ! C4crop


   ! flux leaving the active pool
   MatrixA(:,m,iactive_pool,iactive_pool) = MatrixA(:,m,iactive_pool,iactive_pool) * &
        flux_tot_coeff(2)

   MatrixA(:,m,iactive_pool,islow_pool)= MatrixA(:,m,iactive_pool,islow_pool) * &
        flux_tot_coeff(2)

   MatrixA(:,m,iactive_pool,ipassive_pool) = MatrixA(:,m,iactive_pool,ipassive_pool) * &
        flux_tot_coeff(2)


   MatrixA(:,m,islow_pool,iactive_pool) =  MatrixA(:,m,islow_pool,iactive_pool) * &
        flux_tot_coeff(2)


   ! flux leaving the slow pool
   MatrixA(:,m,islow_pool,islow_pool) = MatrixA(:,m,islow_pool,islow_pool) * &
        flux_tot_coeff(2)
```

```
          MatrixA(:,m,ipassive_pool,iactive_pool) = MatrixA(:,m,ipassive_pool,iactive_pool) * &
                flux_tot_coeff(2)

          MatrixA(:,m,ipassive_pool,islow_pool) = MatrixA(:,m,ipassive_pool,islow_pool) * &
                flux_tot_coeff(2)

          ! flux leaving the passive pool
          MatrixA(:,m,ipassive_pool,ipassive_pool) =  MatrixA(:,m,ipassive_pool,ipassive_pool) * &
                flux_tot_coeff(2)

       ENDIF

       IF (bavard.GE.4) WRITE(numout,*) 'Finish␣to␣fill␣MatrixA'

    ENDDO ! Loop over # PFTS


    ! 4.2 Add Identity for each submatrix(7,7)

    DO j = 1,nbpools
       MatrixA(:,:,j,j) = MatrixA(:,:,j,j) + un
    ENDDO


 ENDIF ! (spinup_analytic)
```

After this step, the part of matrix A corresponding to the exchanges between the three soil pools has been set :

$$\begin{pmatrix} a_{55} & a_{56} & a_{57} \\ a_{65} & a_{66} & 0 \\ a_{75} & 0 & a_{77} \end{pmatrix}$$

5.3.4 lpj_fire

```
  ! litter structural above
  MatrixA(:,j,istructural_above,istructural_above) =
  MatrixA(:,j,istructural_above,istructural_above) - firefrac(:,j) +
  firefrac(:,j)* struc_residual(:)*( 1. - bcfrac(:))

  ! litter_metabolic above
  MatrixA(:,j,imetabolic_above,imetabolic_above) =
  MatrixA(:,j,imetabolic_above,imetabolic_above) - firefrac(:,j)
```

This flux is taken in account for the above litter (metabolic+structural) only if the flag FIRE_DISABLE is set to no. In this case, the elements a_{11} and a_{33} are corrected.

5.3.5 stomate module : the core of the algorithm

matrixA and vectorB are collected by stomate for a update of the saved variables called MatrixV and VectorU. These variables match to the sequences U_n and V_n defined in the section 4. In the code, it gives :

```
  !
  ! 1. Update V and U every time step
  !
  DO m = 2,nvm
     DO j = 1,kjpindex
        ! V <- A * V
        MatrixV(j,m,:,:) = MATMUL(matrixA(j,m,:,:),MatrixV(j,m,:,:))
        ! U <- A*U + B
        VectorU(j,m,:) = MATMUL(matrixA(j,m,:,:),VectorU(j,m,:)) + vectorB(j,m,:)
     ENDDO ! loop pixels
  ENDDO ! loop PFTS
```

matrixA and vectorB are recalculated at the time step of SECHIBA because the subroutines littercalc and soilcarbon are called every 30 minutes.

The system is solved periodically at the end of the year. The period is calculated as a multiple of the number of forcing years. For example, if the run uses one forcing year, the system is solved every ten years. By this way, we can smooth the intrinsic instabilities of the model and evaluate the convergence more easily. Moreover, it reduces the number of resolutions of the system. Some tests showed that the number of resolutions has no impacts on the convergence (because of no feedback of the resolution into the physical processes). This piece of code is the core of the analytical spin-up algorithm :

```fortran
IF(EndOfYear) THEN

   !
   ! 3.1 Increase the years counter every EndOfyear
   !
   global_years = global_years + 1


   !
   ! 3.2 Is global_years is a multiple of the period time ?
   !

   !
   ! 3.2.1 When global_years is a multiple of the period time, we calculate :
   !        1) the mean nbp flux over the period. This value is restarted
   !        2) we solve the matrix system by Gauss Jordan method
   !        3) We test if a point is at equilibrium : if yes, we mark the point (ok_equilibrium array)
   !        4) Then we reset the matrix
   !        5) We erase the carbon_stock calculated by ORCHIDEE by the one found by the method
   IF( MOD(global_years, time_spinup_factor*nbyear) == 0) THEN
      ! The number total of days during the forcing period is given by :
      !     time_spinup_factor*nbyear* 365. (we consider only the noleap calendar)
      nbp_flux(:) = nbp_accu(:) / ( time_spinup_factor*nbyear* 365.)
      ! Reset the values
      nbp_accu(:) = zero

      carbon_stock(:,ibare_sechiba,:) = zero
      ! Prepare the matrix for the resolution
      ! Add a temporary matrix W which contains I-MatrixV
      ! we should take the opposite of matrixV and add the identitiy : we solve (I-MatrixV)*C = VectorU
      MatrixW(:,:,:,:) = moins_un * MatrixV(:,:,:,:)
      DO jv = 1,nbpools
         MatrixW(:,:,jv,jv) =  MatrixW(:,:,jv,jv) + un
      ENDDO
      carbon_stock(:,:,:) = VectorU(:,:,:)

      !
      !  Solve the linear system
      !
      DO m = 2,nvm
         DO j = 1,kjpindex
            ! the solution will be stored in VectorU : so it should be restarted before
            ! loop over npts and nvm, so we solved npts*(nvm-1) (7,7) linear systems
            CALL gauss_jordan_method(nbpools,MatrixW(j,m,:,:),carbon_stock(j,m,:))
         ENDDO
      ENDDO

      ! Reset temporary matrixW
      MatrixW(:,:,:,:) = zero

      previous_stock(:,:,:) = current_stock(:,:,:)
      current_stock(:,:,:) = carbon_stock(:,:,:)
      ! The relative error is calculated over the passive carbon pool (sum over the pfts) over the pixel
      CALL error_L1_passive(kjpindex,nvm, nbpools, current_stock, previous_stock, &
           &                 eps_carbon, carbon_eq)

      !! ok_equilibrium is saved,
      WHERE( carbon_eq(:) .AND. .NOT.(ok_equilibrium(:)) )
         ok_equilibrium(:) = .TRUE.
      ENDWHERE

      ! Reset matrixV for the pixel to the identity matrix and vectorU to zero
      MatrixV(:,:,:,:) = zero
      VectorU(:,:,:) = zero
      DO jv = 1,nbpools
         MatrixV(:,:,jv,jv) = un
```

```
            END DO
            WRITE(numout,*) 'Reset␣for␣matrixV␣and␣VectorU␣done'

            !! Write the values found in the standard outputs of ORCHIDEE
            litter(:,istructural,:,iabove) = carbon_stock(:,:,istructural_above)
            litter(:,istructural,:,ibelow) = carbon_stock(:,:,istructural_below)
            litter(:,imetabolic,:,iabove) = carbon_stock(:,:,imetabolic_above)
            litter(:,imetabolic,:,ibelow) = carbon_stock(:,:,imetabolic_below)
            carbon(:,iactive,:) = carbon_stock(:,:,iactive_pool)
            carbon(:,islow,:) = carbon_stock(:,:,islow_pool)
            carbon(:,ipassive,:) = carbon_stock(:,:,ipassive_pool)

            !! Add for parallelisation
            IF (.NOT. ALLOCATED(ok_equilibrium_g)) THEN
               IF(is_root_prc) THEN
                  ALLOCATE(ok_equilibrium_g(nbp_glo),stat=ier)
                  IF (ier /= 0) THEN
                     WRITE(numout,*) '␣error␣in␣memory␣allocation␣for␣ok_equilibrium_g.'
                     STOP 'stomate_main'
                  ENDIF
               ELSE
                  ALLOCATE(ok_equilibrium_g(0))
               END IF
            END IF ! IF (.NOT. ALLOCATED(ok_equilibrium_g))
            CALL gather(ok_equilibrium,ok_equilibrium_g)

            ! Final step, if all the points are at equilibrium, we can stop the job
            IF(is_root_prc) THEN
               IF(ALL(ok_equilibrium_g)) WRITE(*,*) 'Equilibrium␣for␣carbon␣pools␣is␣reached'
            ENDIF

         ENDIF ! ( MOD(global_years,time_spinup_factor*nbyear) == 0)

      ENDIF ! (EndOfYear)
```

global_years is a global counter of years. It is increased at the end of each year. It helps to check if the condition :

```
( MOD(global_years, time_spinup_factor*nbyear) == 0)
```

is respected. time_spinup_factor is a parameter whose value depends on the forcing period used. global_years is restarted.

nbp_flux is a diagnostic variable representing the daily mean nbp flux over the forcing period. This variable is restarted.

nbp_accu is the cummulated nbp_flux over the forcing period. This variable is restarted but is reset to zero after each period.

carbon_stock contains the solution of the system (the seven carbon pools). It is used as buffer variable for VectorU.

MatrixW is a temporary variable needed to stock the matrix $I - V_n$.

previous_stock and current_stock represent respectively the solution at the previous period and the current period. So we can check graphically the relative error. Both variables are restarted.

The subroutine error_L1_passive in gauss_jordan_method module is called to compute the relative error on the passive pool.

carbon_eq is a temporary logical array which marks the point as true if the criterion over the passive was less than eps_carbon.

ok_equilibrium is a saved logical array which keeps in memory the points which are considered at equilibrium. Once a point has been marked at steady state, it cannot be unmarked.

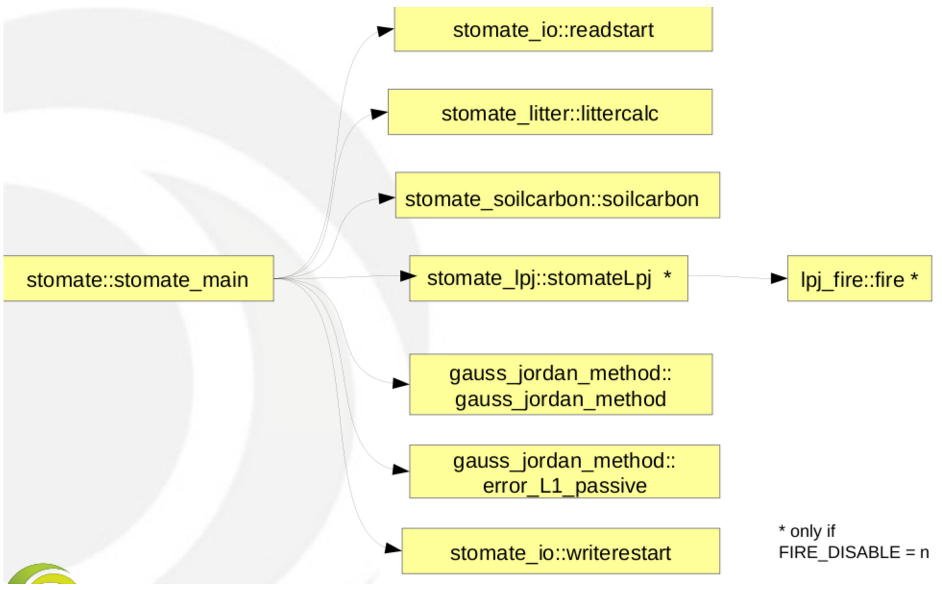
When the resolution is done, we reset MatrixV to identity and VectorU to zero for the next forcing period. After we overwrite the variables litter and carbon used by ORCHIDEE with carbon_stock.

The final test consists to check if all the values of ok_equilibrium (ok_equilibrium_g is used for the parallelization) are set to true. If the test is OK, a sentence is written in the standard output of the processor 0. This sentence can be interpreted by the libIGCM scripts (modified) so the simulation can be stopped automatically before the end.

5.3.6 stomate_io

The analytical method for the spin-up needs to restart the following variables : global_years, ok_equilibrium, nbp_accu, nbp_flux, MatrixV, VectorU, previous_stock, current_stock.

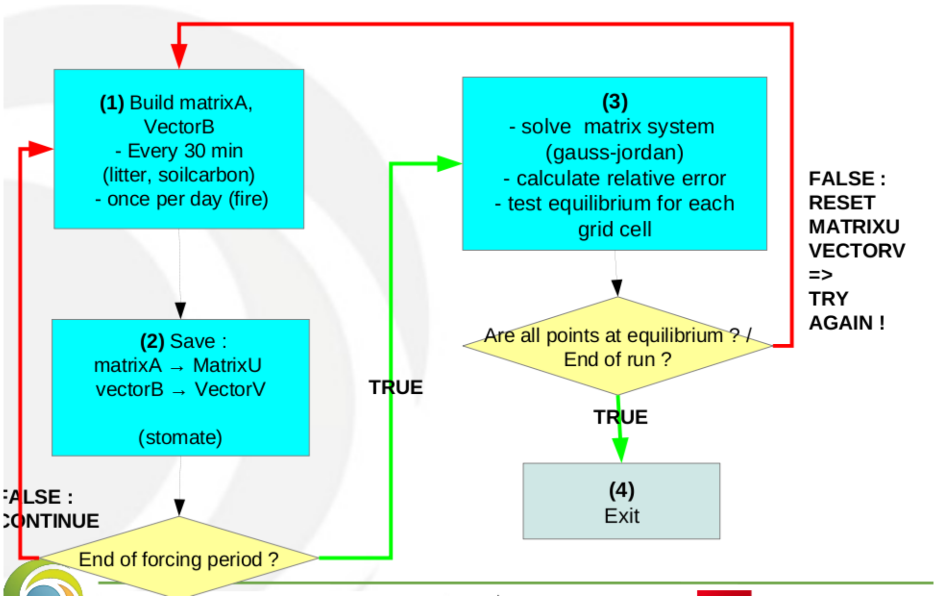
6 Call graph



7 Summary : algorithm

The algorithm implemented in ORCHIDEE can be summarized :

1. Calculation of the periodicity of the resolution in function of the forcing period
2. Calculation of the fluxes matrices matrixA and vectorB for each time step
3. Storage in the variables MatrixV and VectorU over the period
4. Resolution by the Gauss Jordan method
5. computation of the relative error and test


(1) Build matrixA, VectorB - Every 30 min (litter, soilcarbon) - once per day (fire)
(2) Save : matrixA → MatrixU vectorB → VectorV (stomate)
FALSE : CONTINUE
End of forcing period ?
TRUE
(3) - solve matrix system (gauss-jordan) - calculate relative error - test equilibrium for each grid cell
Are all points at equilibrium ? / End of run ?
FALSE : RESET MATRIXU VECTORV => TRY AGAIN !
TRUE
(4) Exit

8 Evaluation of the steady state

For evaluating the steady state, we have chosen two criterions : the relative error on the passive pool and the nbp. The relative error on the passive pool is evaluated via the following formula :

$$\frac{100 \cdot |C^{n}_{pass} - C^{n-1}_{pass}|}{|C^{n-1}_{pass}|} < \varepsilon \tag{9}$$

where ε (called eps_carbon in the code) can be defined by the user. By defaut, eps_carbon worths 0.01%.

The nbp is accumulated over the forcing period by (variable nbp_accu). When the linear systems are solved, the daily mean nbp is calculated over the forcing period by the variable nbp_flux (look at stomate module in the previous section).

```
IF (spinup_analytic) THEN
   nbp_accu(:) = nbp_accu(:) + (-SUM(co2_flux_daily(:,:) * veget_max(:,:),dim=2) - &
                 (convflux(:) + cflux_prod10(:) + &
                 cflux_prod100(:)) - harvest_above(:))/1e3
ENDIF
```

9 Advantages and incovenients of the method

Many advantages :

1. fully integrated in ORCHIDEE ⇒ more stable method.
2. faster as ORCHIDEE + FORCESOIL configuration.
3. easy to parallelize -(OpenMP + MPI).
4. no need of any forcing file like stomate_Cforcing.nc or subprogram like forcesoil ⇒ simplification of the code.
5. extension to nitrogen pools.

6. possible coupling with TESTSTOMATE.
7. reasonable complexity.
8. easy to evaluate the steady state (error + nbp).
9. compatible with the externalization.
10. first step to a common protocol for the spin-up inside the ORCHIDEE users.

Few incoveniients :

1. no formal proof of the positivity of the solution of the system
2. calculated once a year : no carbon inter annual variability. An other method using Fourier series has been developped : see references.
3. not modular

10 Protocol

Before starting a spin-up run, it is advised to choose noleap focing files. The following protocol is valid only if you use libIGCM and the SPINUP configuration. The SPINUP script will finally only use OOL_SEC_STO configuration and does not need FORCESOIL or TESTSTOMATE.

1. Go to config/ORCHIDEE_OL directory.
2. Go to OOL_SEC_STO configuration and check in COMP/stomate.card that SPINUP_ANALYTIC is set to true.
3. Get out and go now inside SPINUP directory.
4. Edit and modify the config card.
5. Edit spinup.card in COMP directory and check if OK_CO2, OK_STOMATE are set to true. The algebraic method can work with IMPOSE_VEG or LAND_USE activated. Set FORCESOIL_NB_YEAR at the right value. This parameter corresponds to the length of your forcing period. Check the paths to your forcing files.
6. Calculate your number of iterations. For example, if you intend to run 300 years with a 10Y forcing files, you have to set **duree_initstomate** to 10, then set **n_iter** to 28 with **duree_sechiba** to 10 (the rest is zero) and **duree_final** to 10. Here is an extract of spinup.card :

```
[UserChoices]

# Should we start the analytic resolution of the spinup?
SPINUP_ANALYTIC=y

#
###— STOMATE flag
#
ok_stomate=y

#
###— CO2 flag
#
ok_co2=y

#
###— historical vegetation maps
#
land_use=y

    # SPINUP configuration :
    # ————————————————
# Initialisation for spin-up :
# sechiba alone (!!! only if ok_stomate == n !!!)
duree_nostomate=0
# sechiba and stomate
duree_inistomate=10
# teststomate (only if duree_nostomate or duree_inistomate > 0)
duree_offlineini=0

# Loop configuration for spin-up :
# The whole job is restarted n_iter times
n_iter=28
# orchidee with sechiba (and stomate if ok_stomate=y below)
duree_sechiba=10
# teststomate
duree_stomate=0
# forcesoil
duree_carbonsol=0

# Finalization for spin-up :
# all orchidee
duree_final=10
# This last parameter must be non-zero.

[SubJobForcingFile]
# Boundary Files for ORCHIDEE subjobs :
List= (/home/orchidee03/dsolyga/ncc_for_${year}.nc, forcing_file.nc)
#List= (${R_BC}/OOL/${config_UserChoices_TagName}/CRU/force${year}.nc, .)
ListNonDel= ()

[SubJobParams]
# You can specify here any parameters to be modified in sechiba.def, stomate.def or driver.def for Sp
# NEW : due to split of orchidee.def in component specific parameter files ,
#        you must add here a prefix for the specific parameter file.
driver_DEBUG_INFO=n
sechiba_LONGPRINT=n
stomate_FORCESOIL_NB_YEAR=10
```

7. Change the value of the parameter EPS_CARBON if needed in PARAM directory. This parameter should be in run.def or stomate.def according the version. Do not forget that this parameter is set to 0,01% by default.
8. Create your job and launch your run.

If you do not use libIGCM, you can obviously activate this method. The libIGCM interface is more convenient to use wih the SPINUP configuration. The scripts lets us to have a kind of automatic job which stops when equilibrium has been reached before the end.

11 References

- Lardy, R., et al., A new method to determine soil organic carbon equilibrium, Environmental Modelling Software (2011), doi :10.1016/j.envsoft.2011.05.016

- Thornton, P.E., Rosenbloom, N.A., 2005. Ecosystem model spin-up : estimating steady state conditions in a coupled terrestrial carbon and nitrogen cycle model. Ecological Modelling 189, 25e48

- J. Xia et al., A semi-analytical solution to accelerate spin-up of a coupled carbon and nitrogen land model to steady state, Geosci. Model Dev. Discuss., 5, 803–836, 2012

- Martin, P.M., Cordier, S., Balesdent, J., Arrouays, D., 2007. Periodic solutions for soil carbon dynamics equilibriums with time-varying forcing variables. Ecological Modelling 204, 523e530.

12 Annex A : mathematical proof

In this section we will prove the following statement by using the mathematical induction :

$$\forall n \geqslant 1, C_n = \sum_{j=0}^{n-1}(\prod_{k=j+1}^{n-1} D_k)E_j + (\prod_{j=0}^{n-1} D_j)C_0 \qquad (*)$$

For $n = 1$, we have :

$$C_1 = D_0 \cdot C_0 + E_0$$

because $\prod_{k=1}^{0} D_k) = 1$ by convention of the empty product.

Let us suppose the assumption true for all $m \leqslant n$. We have :

$$\begin{aligned} C_{n+1} &= D_n \cdot C_n + E_n \\ &= D_n \cdot \left(\sum_{j=0}^{n-1}(\prod_{k=j+1}^{n-1} D_k)E_j + (\prod_{j=0}^{n-1} D_j)C_0 \right) + E_n \\ &= \sum_{j=0}^{n-1}(\prod_{k=j+1}^{n} D_k)E_j + E_n + (\prod_{j=0}^{n} D_j) \\ &= \sum_{j=0}^{n}(\prod_{k=j+1}^{n} D_k)E_j + (\prod_{j=0}^{n} D_j) \end{aligned}$$

because $E_n = (\prod_{j=n+1}^{n} D_j) \cdot E_n$. By the principle of mathematical, $(*)$ is true.