

# **Matrix approach to land carbon cycle modeling: A case study with Community Land Model**

Yuanyuan Huang<sup>1</sup>, Xingjie Lu<sup>1</sup>, Zheng Shi, David Lawrence, Charles Koven, Jianyang Xia, Zhenggang Du, Erik Kluzek, Yiqi Luo

## **Supplementary Information**

### **Demonstration code**

The demonstration code reads input data from the supplementary zip file which contains one netcdf file that provides required input variables for the matrix simulation and one mat file about soil depth information. Supplementary data are available at <https://doi.org/10.1594/PANGAEA.881568>. The matlab code runs forward simulation for 12 steps (12 months) from the initial condition provided by the input netcdf file. The purpose is mainly to document how matrices are generated and applications are not limited to the forward simulation demonstrated here.

Corresponding to: [yuanyuanhuang2011@gmail.com](mailto:yuanyuanhuang2011@gmail.com); [Xingjie.Lu@nau.edu](mailto:Xingjie.Lu@nau.edu)

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Demo global simulation; 12 timesteps
% matrix representation of CLM45, 7 pools;
% CWDC; Metabolic Litter; Cellulose Litter; Lignin Litter
% Fast SOC; active SOC; slow SOC; passive SOC
% Yuanyuan Huang contact: yuanyuanhuang2011@gmail.com
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
close all;
format long e;

use_vertsoilc = 1;
nspools = 7; % number of pools if no vertical; can be used if interested in the non-vertical
version
nspools_vr = 70; % number of pools if vertical
nlevdecomp = 10; % number of soil layers

load soildepth.mat; %soil depth information
file_in = 'global_demo_in.nc'; % input file
wscalar_in = ncread(file_in, 'W_SCALAR'); % moisture scalar
tscalar_in = ncread(file_in, 'T_SCALAR'); % temperature scalar
oscalar_in = ncread(file_in, 'O_SCALAR'); % oxygen scalar
fpi_vr_in = ncread(file_in, 'FPI_VR'); % nitrogen scalar

cwdc_vr_in = ncread(file_in, 'CWDC_VR'); % initial value for pools
litr1c_vr_in = ncread(file_in, 'LITR1C_VR');
litr2c_vr_in = ncread(file_in, 'LITR2C_VR');
litr3c_vr_in = ncread(file_in, 'LITR3C_VR');
soil1c_vr_in = ncread(file_in, 'SOIL1C_VR');
soil2c_vr_in = ncread(file_in, 'SOIL2C_VR');
soil3c_vr_in = ncread(file_in, 'SOIL3C_VR');

altmax_in = ncread(file_in, 'ALTMAX');
altmax_lastyear_in = ncread(file_in, 'ALTMAX_LASTYEAR');
cellsand_in = ncread(file_in, 'CELLSAND');

mlon_in = ncread(file_in, 'LON'); % longitude
mlat_in = ncread(file_in, 'LAT'); % latitude

totc_to_cwdc_in = ncread(file_in, 'TOTC2CWDC_VR'); %litter input gc/m2/s
totc_to_litrmet_in = ncread(file_in, 'TOTC2LITRMETC_VR');
totc_to_litr cel_in = ncread(file_in, 'TOTC2LITRCELC_VR');
totc_to_litr lig_in = ncread(file_in, 'TOTC2LITRLIGC_VR');

secsday = 86400;
dt = secsday*30; % suppose it is monthly simulation
nstep = length(oscalar_in(1,1,1,:));
lon_in = length(oscalar_in(:,1,1,1));
lat_in = length(oscalar_in(1,:,1,1));

totc_layer = totc_to_cwdc_in(:, :, 1:10, :) + totc_to_litrmet_in(:, :, 1:10, :) + ...
    totc_to_litr cel_in(:, :, 1:10, :) + totc_to_litr lig_in(:, :, 1:10, :);
totc = sum(totc_layer, 3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% calculate allocation (B) of litter input
for lon = 1:lon_in
    %for lon = 119:119
    for lat = 1:lat_in
        %for lat = 44:44
        for n = 1:nstep
            if (totc(lon, lat, 1, n) < 1e-36)
                b_cwdc(lon, lat, :, n) = 0.0;
                b_litrmet(lon, lat, :, n) = 0.0;
                b_litr cel(lon, lat, :, n) = 0.0;
            end
        end
    end
end

```

```

..... b_litrllig(lon,lat,:,n) = 0.0;
..... else
..... b_cwdc(lon,lat,:,n) = totc_to_cwdc_in(lon,lat,:,n)/totc(lon,lat,1,n);
..... b_litrmet(lon,lat,:,n) = totc_to_litrmet_in(lon,lat,:,n)/totc(lon,lat,1,n);
..... b_litrcll(lon,lat,:,n) = totc_to_litrcll_in(lon,lat,:,n)/totc(lon,lat,1,n);
..... b_litrllig(lon,lat,:,n) = totc_to_litrllig_in(lon,lat,:,n)/totc(lon,lat,1,n);
..... end
..... end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
cmatrix_grid_tot = NaN(lon_in, lat_in, nstep);
for lon = 1:lon_in %for lon = 119:119
  for lat = 1:lat_in %for lat = 44:44
    %%% initialization
    cmatrix_cpools(1,:,1)=litr1c_vr_in(lon,lat,1:nlevdecomp,1);
    cmatrix_cpools(1,:,2)=litr2c_vr_in(lon,lat,1:nlevdecomp,1);
    cmatrix_cpools(1,:,3)=litr3c_vr_in(lon,lat,1:nlevdecomp,1);
    cmatrix_cpools(1,:,4)=cwdc_vr_in(lon,lat,1:nlevdecomp,1);
    cmatrix_cpools(1,:,5)=soillc_vr_in(lon,lat,1:nlevdecomp,1);
    cmatrix_cpools(1,:,6)=soil2c_vr_in(lon,lat,1:nlevdecomp,1);
    cmatrix_cpools(1,:,7)=soil3c_vr_in(lon,lat,1:nlevdecomp,1);
    .....
    for ts = 1:nstep
      totc_to_cwd(ts,:) = totc_to_cwdc_in(lon,lat,1:nlevdecomp,ts);
      totc_to_litrmet(ts,:) = totc_to_litrmet_in(lon,lat,1:nlevdecomp,ts);
      totc_to_litrcll(ts,:) = totc_to_litrcll_in(lon,lat,1:nlevdecomp,ts);
      totc_to_litrllig(ts,:) = totc_to_litrllig_in(lon,lat,1:nlevdecomp,ts);
      .....
      b_to_cwd(ts,:) = b_cwdc(lon,lat,1:nlevdecomp,ts);
      b_to_litrmet(ts,:) = b_litrmet(lon,lat,1:nlevdecomp,ts);
      b_to_litrcll(ts,:) = b_litrcll(lon,lat,1:nlevdecomp,ts);
      b_to_litrllig(ts,:) = b_litrllig(lon,lat,1:nlevdecomp,ts);
      .....
      totcm(ts,1) = totc(lon,lat,1,ts);
      .....
    end
    for ts = 1:nstep-1
      %==== vertical matrix
      tri_ma = tri_matrix(use_vertsoilc,altmax_in(lon,lat,ts),...
        altmax_lastyear_in(lon,lat,ts));
      %==== transfer matrix
      a_ma = a_matrix(use_vertsoilc,cellsand_in(lon,lat,1:nlevdecomp,ts));
      %==== K * Scalar matrix
      kk_ma =
        kk_matrix(use_vertsoilc,wscllar_in(lon,lat,1:nlevdecomp,ts),tscllar_in(lon,lat,1:nlev
          decomp,ts),...
        oscllar_in(lon,lat,1:nlevdecomp,ts),fpi_vr_in(lon,lat,1:nlevdecomp,ts));
      cmatrix_in = zeros(1,70);
      cmatrix_in_1 = zeros(1,7);
      allo_in = zeros(1,70);
      if (use_vertsoilc)
        for j = 1:nlevdecomp
          cmatrix_current(j) = cmatrix_cpools(ts,j,4); % cwdc
          cmatrix_current(j+1*nlevdecomp) = cmatrix_cpools(ts,j,1); % litr1c
          cmatrix_current(j+2*nlevdecomp) = cmatrix_cpools(ts,j,2); % litr2c
          cmatrix_current(j+3*nlevdecomp) = cmatrix_cpools(ts,j,3); % litr3c
          cmatrix_current(j+4*nlevdecomp) = cmatrix_cpools(ts,j,5); % soillc
          cmatrix_current(j+5*nlevdecomp) = cmatrix_cpools(ts,j,6); % soil2c
          cmatrix_current(j+6*nlevdecomp) = cmatrix_cpools(ts,j,7); % soil3c
          % input to cwd, litr
          cmatrix_in(j) = totc_to_cwd(ts,j) * dt;
          cmatrix_in(j+1*nlevdecomp) = totc_to_litrmet(ts,j) * dt;

```

```

..... cmatrix_in(j+2*nlevdecomp) = totc_to_litr cel(ts,j) * dt;
..... cmatrix_in(j+3*nlevdecomp) = totc_to_litr lig(ts,j) * dt;
.....
..... allo_in(j) = b_to_cwd(ts,j);
..... allo_in(j+1*nlevdecomp) = b_to_litr met(ts,j);
..... allo_in(j+2*nlevdecomp) = b_to_litr cel(ts,j);
..... allo_in(j+3*nlevdecomp) = b_to_litr lig(ts,j);
.....
..... end %end of decay matrix * scalar matrix

..... totc_in = totcm(ts,1)*dt;
..... emu_in = allo_in' * totc_in;
..... %=====
..... % the main part update state variables
..... cmatrix_next = cmatrix_current' + emu_in + ...
..... (a_ma*kk_ma-tri_ma)*cmatrix_current'*dt;
..... %=====
..... tnext = ts +1;
..... for j = 1:nlevdecomp
..... cmatrix_cpools(tnext,j,4) = cmatrix_next(j);
..... cmatrix_cpools(tnext,j,1) = cmatrix_next(j+1*nlevdecomp);
..... cmatrix_cpools(tnext,j,2) = cmatrix_next(j+2*nlevdecomp);
..... cmatrix_cpools(tnext,j,3) = cmatrix_next(j+3*nlevdecomp);
..... cmatrix_cpools(tnext,j,5) = cmatrix_next(j+4*nlevdecomp);
..... cmatrix_cpools(tnext,j,6) = cmatrix_next(j+5*nlevdecomp);
..... cmatrix_cpools(tnext,j,7) = cmatrix_next(j+6*nlevdecomp);
..... end
.....
..... else
% ..... cmatrix_current_1(1) = cmatrix_cpools(ts,1,4);
% ..... cmatrix_current_1(2) = cmatrix_cpools(ts,1,1);
% ..... cmatrix_current_1(3) = cmatrix_cpools(ts,1,2);
% ..... cmatrix_current_1(4) = cmatrix_cpools(ts,1,3);
% ..... cmatrix_current_1(5) = cmatrix_cpools(ts,1,5);
% ..... cmatrix_current_1(6) = cmatrix_cpools(ts,1,6);
% ..... cmatrix_current_1(7) = cmatrix_cpools(ts,1,7);
% ..... cmatrix_in_1(1) = totc_to_cwd(ts,1) * dt;
% ..... cmatrix_in_1(2) = totc_to_litr met(ts,1) * dt;
% ..... cmatrix_in_1(3) = totc_to_litr cel(ts,1) * dt;
% ..... cmatrix_in_1(4) = totc_to_litr lig(ts,1) * dt;
% .....
% ..... cmatrix_next_1 = cmatrix_current_1' + cmatrix_in_1' + ...
% ..... a_ma*kk_ma*cmatrix_current_1'*dt;
% ..... tnext = ts +1;
% ..... cmatrix_cpools(tnext,1,4) = cmatrix_next_1(1,1);
% ..... cmatrix_cpools(tnext,1,1) = cmatrix_next_1(2,1);
% ..... cmatrix_cpools(tnext,1,2) = cmatrix_next_1(3,1);
% ..... cmatrix_cpools(tnext,1,3) = cmatrix_next_1(4,1);
% ..... cmatrix_cpools(tnext,1,5) = cmatrix_next_1(5,1);
% ..... cmatrix_cpools(tnext,1,6) = cmatrix_next_1(6,1);
% ..... cmatrix_cpools(tnext,1,7) = cmatrix_next_1(7,1);
..... end %end of decay matrix + scalar matrix without vertical
..... end % end of time steps
..... %===== vertically integrated total carbon=====
..... cmatrix_grid_tot(lon,lat,:) = sum(cmatrix_cpools,3)*dz(1:nlevdecomp);
..... end
end

%write_nc_test('demo',lon_in,lat_in,nstep,m lon_in,m lat_in,...
% ..... cmatrix_grid_tot); %save simulation result

% end of the main fuction

```

```

%*****
% function to calculate the transfer matrix (A matrix) ..... *****
%*****

%clear all;
%close all;
function a_out = a_matrix(use_vertsoilc,cellsand_in)
nlevdecomp = 10;
cellsand = cellsand_in;
%cellsand = [0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2];
nspools = 7;
nspools_vr = 70;
%use_vertsoilc = 1;
a_ma_vr = zeros(nspools_vr,nspools_vr);
a_ma = zeros(nspools,nspools);

for j = 1:nlevdecomp
    t = 0.85 - 0.68 * 0.01 * (100 - cellsand(j));
    f_sls2(j) = 1 - 0.004 / (1 - t);
    f_sls3(j) = 0.004 / (1 - t);
    rf_sls2(j) = t;
    rf_sls3(j) = t;
end

% set path fractions
f_s2s1 = 0.42/(0.45);
f_s2s3 = 0.03/(0.45);

if (use_vertsoilc)
    for j = 1:nspools_vr
        a_ma_vr(j,j) = -1.0;
    end
    for j = 1:nlevdecomp
        % respiration fraction
        % 1,1,1,fsls2,fsls3,fs2s1,fs2s3,1, cwd_fcel,cwd_flg
        pathfrac_decomp_cascade = [1,1,1,f_sls2(j),f_sls3(j),f_s2s1,f_s2s3,1,0.76,0.24];
        % transfer
        % rf_lls1,l2s1,l3s2,sls2,sls3,s2s1,s2s3,s3s1,cwdl2,cwdl3
        rf_decomp_cascade = [0.55,0.5,0.5,rf_sls2(j),rf_sls3(j),0.55,0.55,0.55,0,0];
        a_ma_vr((3-1)*nlevdecomp+j,(1-1)*nlevdecomp+j) =
            (1.0-rf_decomp_cascade(9))*pathfrac_decomp_cascade(9);
        a_ma_vr((4-1)*nlevdecomp+j,(1-1)*nlevdecomp+j) =
            (1.0-rf_decomp_cascade(10))*pathfrac_decomp_cascade(10);
        a_ma_vr((5-1)*nlevdecomp+j,(2-1)*nlevdecomp+j) =
            (1.0-rf_decomp_cascade(1))*pathfrac_decomp_cascade(1);
        a_ma_vr((5-1)*nlevdecomp+j,(3-1)*nlevdecomp+j) =
            (1.0-rf_decomp_cascade(2))*pathfrac_decomp_cascade(2);
        a_ma_vr((5-1)*nlevdecomp+j,(6-1)*nlevdecomp+j) =
            (1.0-rf_decomp_cascade(6))*pathfrac_decomp_cascade(6);
        a_ma_vr((5-1)*nlevdecomp+j,(7-1)*nlevdecomp+j) =
            (1.0-rf_decomp_cascade(8))*pathfrac_decomp_cascade(8);
        a_ma_vr((6-1)*nlevdecomp+j,(4-1)*nlevdecomp+j) =
            (1.0-rf_decomp_cascade(3))*pathfrac_decomp_cascade(3);
        a_ma_vr((6-1)*nlevdecomp+j,(5-1)*nlevdecomp+j) =
            (1.0-rf_decomp_cascade(4))*pathfrac_decomp_cascade(4);
        a_ma_vr((7-1)*nlevdecomp+j,(5-1)*nlevdecomp+j) =
            (1.0-rf_decomp_cascade(5))*pathfrac_decomp_cascade(5);
        a_ma_vr((7-1)*nlevdecomp+j,(6-1)*nlevdecomp+j) =
            (1.0-rf_decomp_cascade(7))*pathfrac_decomp_cascade(7);
    end
    a_out = a_ma_vr;
else
    for j = 1:nspools
        a_ma(j,j) = -1.0;
    end
end

```

```

pathfrac_decomp_cascade=[1,1,1,f_sls2(1),f_sls3(1),f_s2s1,f_s2s3,1,0.76,0.24];
rf_decomp_cascade=[0.55,0.5,0.5,rf_sls2(1),rf_sls3(1),0.55,0.55,0.55,0,0];
a_ma(3,1)=(1.0-rf_decomp_cascade(9))*pathfrac_decomp_cascade(9);
a_ma(4,1)=(1.0-rf_decomp_cascade(10))*pathfrac_decomp_cascade(10);
a_ma(5,2)=(1.0-rf_decomp_cascade(1))*pathfrac_decomp_cascade(1);
a_ma(5,3)=(1.0-rf_decomp_cascade(2))*pathfrac_decomp_cascade(2);
a_ma(5,6)=(1.0-rf_decomp_cascade(6))*pathfrac_decomp_cascade(6);
a_ma(5,7)=(1.0-rf_decomp_cascade(8))*pathfrac_decomp_cascade(8);
a_ma(6,4)=(1.0-rf_decomp_cascade(3))*pathfrac_decomp_cascade(3);
a_ma(6,5)=(1.0-rf_decomp_cascade(4))*pathfrac_decomp_cascade(4);
a_ma(7,5)=(1.0-rf_decomp_cascade(5))*pathfrac_decomp_cascade(5);
a_ma(7,6)=(1.0-rf_decomp_cascade(7))*pathfrac_decomp_cascade(7);
a_out=a_ma;
end %!!!!!!!!!!!!!! end of transfer matrix !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
end
%*****

%*****
%function to calculalte K multiply scalar matrix *****
%*****
%clear all;
%close all;
function kk_out=kk_matrix(use_vertsoilc,wscalar_in,tscalar_in,oscalar_in,...
    fpi_vr_in)
nlevdecomp=10;
days_per_year=365;
secsday=86400;
%use_vertsoilc=1;
load soildepth.mat;

%note scalars change with time
n_scalar=fpi_vr_in;%nitrogen, fpi
t_scalar=tscalar_in;%temperature
w_scalar=wscalar_in;%water
o_scalar=oscalar_in;%oxygen

%turnover rate and time. Copied from SoilBiogeochemDecompCascadeBGCMMod
%!!!!!! the belowground parameters from century
tau_l1=1./18.5;
tau_l2_l3=1./4.9;
tau_s1=1./7.3;
tau_s2=1./0.2;
tau_s3=1./0.0045;
%century leaves wood decomposition rates open, within range of 0--0.5 yr^-1
tau_cwd=1./0.3;
%translate to per-second time constant
k_l1=1./ (secsday * days_per_year * tau_l1);
k_l2_l3=1./ (secsday * days_per_year * tau_l2_l3);
k_s1=1./ (secsday * days_per_year * tau_s1);
k_s2=1./ (secsday * days_per_year * tau_s2);
k_s3=1./ (secsday * days_per_year * tau_s3);
k_frag=1./ (secsday * days_per_year * tau_cwd);

decomp_depth_efolding=0.5;

kk_ma=0.0;

if (use_vertsoilc)
    kk_ma_vr=0.0;%kk_matrix, decay matrix * scalar matrix
    for j=1:nlevdecomp
        depth_scalar(j)=exp(-zsoi(j)/decomp_depth_efolding);
        two_scalar(j)=t_scalar(j)*w_scalar(j)*o_scalar(j);
        kk_ma_vr(j,j)=k_frag*two_scalar(j)*depth_scalar(j);
    end
end

```

```

kk_ma_vr(1*nlevdecomp+j,1*nlevdecomp+j) = k_l1 * two_scalar(j) * depth_scalar(j)*
    n_scalar(j);
kk_ma_vr(2*nlevdecomp+j,2*nlevdecomp+j) = k_l2_l3 * two_scalar(j) * depth_scalar(j)*
    n_scalar(j);
kk_ma_vr(3*nlevdecomp+j,3*nlevdecomp+j) = k_l2_l3 * two_scalar(j) * depth_scalar(j)*
    n_scalar(j);
kk_ma_vr(4*nlevdecomp+j,4*nlevdecomp+j) = k_s1 * two_scalar(j) * depth_scalar(j);
kk_ma_vr(5*nlevdecomp+j,5*nlevdecomp+j) = k_s2 * two_scalar(j) * depth_scalar(j);
kk_ma_vr(6*nlevdecomp+j,6*nlevdecomp+j) = k_s3 * two_scalar(j) * depth_scalar(j);
end
kk_out = kk_ma_vr;
else
two_scalar(1) = t_scalar(1)*w_scalar(1)*o_scalar(1);
kk_ma(1,1) = k_frag * two_scalar(1);
kk_ma(2,2) = k_l1 * two_scalar(1)* n_scalar(1);
kk_ma(3,3) = k_l2_l3 * two_scalar(1)* n_scalar(1);
kk_ma(4,4) = k_l2_l3 * two_scalar(1)* n_scalar(1);
kk_ma(5,5) = k_s1 * two_scalar(1);
kk_ma(6,6) = k_s2 * two_scalar(1);
kk_ma(7,7) = k_s3 * two_scalar(1);
kk_out = kk_ma;
end
end

%*****
% function to calcualte vertical mixing matrix *****
%*****
%clear all;
%close all;
function tri_out = tri_matrix(use_vertsoilc,altmax_in,altmax_lastyear_in)
load soildepth.mat; %dz the same as dzsoi_decomp
nspools = 7;
secsday = 86400;

% use_vertsoilc = 1;
% altmax_in = 35;
% altmax_lastyear_in = 35;

% A function from Patankar, Table 5.2, pg 95
% [m^2/sec] = 5 cm^2 / yr = 1m^2 / 200 yr
som_diffus = 1e-4 / (secsday * 365);
som_adv_flux = 0;
cryoturb_diffusion_k = 5e-4 / (secsday * 365);

max_depth_cryoturb = 3;
max_altdepth_cryoturbation = 2;
nlevdecomp = 10;
nbedrock = 10;

altmax = altmax_in;
altmax_lastyear = altmax_lastyear_in;
%use_vertsoilc = 1;

aaa = @(pe) max(0, (1 - 0.1 * abs(pe))^5);

% Set the distance between the node and the one ABOVE it
dz_node(1) = zsoi(1);
for j = 2:nlevdecomp+1
    dz_node(j) = zsoi(j) - zsoi(j-1);
end

%%
if ((max(altmax, altmax_lastyear) <= max_altdepth_cryoturbation) && ...

```

```

.....(max(altmax, altmax_lastyear) > 0)
% use mixing profile modified slightly from Koven et al. (2009): constant through active layer,
linear decrease from base of active layer to zero at a fixed depth
for j = 1:(nlevdecomp+1)
    if (j <= nbedrock+1)
        if (zsoi(j) < max(altmax, altmax_lastyear))
            diffus(j) = cryoturb_diffusion_k;
            adv_flux(j) = 0;
        else
            diffus(j) = max(cryoturb_diffusion_k * .....
                (1 - (zsoi(j) - max(altmax, altmax_lastyear)) / .....
                (min(max_depth_cryoturb, zsoi(nbedrock+1)) - max(altmax, altmax_lastyear))), 0); %
%go linearly to zero between ALT and max_depth_cryoturb
            adv_flux(j) = 0;
        end
    else
        adv_flux(j) = 0;
        diffus(j) = 0;
    end
end
elseif (max(altmax, altmax_lastyear) > 0)
%constant advection, constant diffusion
for j = 1:(nlevdecomp+1)
    if (j <= nbedrock+1)
        adv_flux(j) = som_adv_flux;
        diffus(j) = som_diffus;
    else
        adv_flux(j) = 0;
        diffus(j) = 0;
    end
end
else
%completely frozen soils--no mixing
for j = 1:(nlevdecomp+1)
    adv_flux(j) = 0;
    diffus(j) = 0;
end
end

%%
for j = 1:(nlevdecomp+1)
    if (j == 1)
        d_m1_zm1(j) = 0;
        w_p1 = (zsoi(j+1) - zsoi(j)) / dz_node(j+1);
        if (diffus(j+1) > 0 && diffus(j) > 0);
            d_p1 = 1 / ((1 - w_p1) / diffus(j) + w_p1 / diffus(j+1)); % Harmonic mean of diffus
        else
            d_p1 = 0;
        end
        d_p1_zp1(j) = d_p1 / dz_node(j+1);
        f_m1(j) = adv_flux(j); % Include infiltration here
        f_p1(j) = adv_flux(j+1);
        pe_m1(j) = 0;
        pe_p1(j) = f_p1(j) / d_p1_zp1(j); % Peclet #
    elseif (j >= nbedrock+1)
        % At the bottom, assume no gradient in d_z (i.e., they're the same)
        w_m1 = (zsoi(j-1) - zsoi(j-1)) / dz_node(j);
        if (diffus(j) > 0 && diffus(j-1) > 0)
            d_m1 = 1 / ((1 - w_m1) / diffus(j) + w_m1 / diffus(j-1)); % Harmonic mean of diffus
        else
            d_m1 = 0;
        end
        d_m1_zm1(j) = d_m1 / dz_node(j);
        d_p1_zp1(j) = d_m1_zm1(j); % Set to be the same
    end
end

```



```

f_m1(j) = adv_flux(j);
%f_p1(j) = adv_flux(j+1)
f_p1(j) = 0;
pe_m1(j) = f_m1(j) / d_m1_zm1(j); % Peclet #
pe_p1(j) = f_p1(j) / d_p1_zp1(j); % Peclet #
else
%Use distance from j-1 node to interface with j divided by distance between nodes
w_m1 = (zsoi(j-1) - zsoi(j-1)) / dz_node(j);
if (diffus(j-1) > 0 && diffus(j) > 0)
d_m1 = 1 / ((1 - w_m1) / diffus(j) + w_m1 / diffus(j-1)); % Harmonic mean of diffus
else
d_m1 = 0;
end
w_p1 = (zsoi(j+1) - zsoi(j)) / dz_node(j+1);
if (diffus(j+1) > 0 && diffus(j) > 0)
d_p1 = 1 / ((1 - w_p1) / diffus(j) + w_p1 / diffus(j+1)); % Harmonic mean of diffus
else
d_p1 = (1 - w_m1) * diffus(j) + w_p1 * diffus(j+1); % Arithmetic mean of diffus
end
d_m1_zm1(j) = d_m1 / dz_node(j);
d_p1_zp1(j) = d_p1 / dz_node(j+1);
f_m1(j) = adv_flux(j);
f_p1(j) = adv_flux(j+1);
pe_m1(j) = f_m1(j) / d_m1_zm1(j); % Peclet #
pe_p1(j) = f_p1(j) / d_p1_zp1(j); % Peclet #
end
end
%%
% Calculate the tridiagonal coefficients
for j = 1:(nlevdecomp+1)
if (j == 1)
a_tri_e(j) = -(d_m1_zm1(j) * aaa(pe_m1(j)) + max(f_m1(j), 0)); % Eqn 5.47 Patankar
c_tri_e(j) = -(d_p1_zp1(j) * aaa(pe_p1(j)) + max(-f_p1(j), 0));
b_tri_e(j) = -a_tri_e(j) - c_tri_e(j);
elseif (j < nlevdecomp+1)
a_tri_e(j) = -(d_m1_zm1(j) * aaa(pe_m1(j)) + max(f_m1(j), 0)); % Eqn 5.47 Patankar
c_tri_e(j) = -(d_p1_zp1(j) * aaa(pe_p1(j)) + max(-f_p1(j), 0));
b_tri_e(j) = -a_tri_e(j) - c_tri_e(j);
else % j==nlevdecomp+1; 0 concentration gradient at bottom
a_tri_e(j) = -1;
b_tri_e(j) = 1;
c_tri_e(j) = 0;
end
end % j; nlevdecomp

%%
tri_ma_vr = 0.0;

if (use_vertsoilc)
for i = 1:nspools
if (i > 1) % vertical transfer only for non-cwdc >—>—>—>
for j = 1:nlevdecomp
tri_ma_vr(j+(i-1)*nlevdecomp, j+(i-1)*nlevdecomp) = b_tri_e(j);
if (j == 1) % upper boundary
tri_ma_vr(1+(i-1)*nlevdecomp, 1+(i-1)*nlevdecomp) = -c_tri_e(1);
end
if (j == 10) % bottom boundary
tri_ma_vr(10+(i-1)*nlevdecomp, 10+(i-1)*nlevdecomp) = -a_tri_e(10);
end
if (j < 10) % avoid tranfer from for example, litr3_10th layer to soil1_1st layer
tri_ma_vr(j+(i-1)*nlevdecomp, j+1+(i-1)*nlevdecomp) = c_tri_e(j);
end
if (j > 1) % avoid tranfer from for example, soil1_1st layer to litr3_10th layer

```

```

.....tri_ma_vr(j+(i-1)*nlevdecomp,j-1+(i-1)*nlevdecomp) = a_tri_e(j);
.....end
.....tri_ma_vr(j+(i-1)*nlevdecomp,:) = tri_ma_vr(j+(i-1)*nlevdecomp,:)/dz(j); %
.....match unit g/m3 Tri/dz
.....end
.....end
end %end of tridiagonal matrix
end
if (use_vertsoilc)
.....tri_out = tri_ma_vr;
else
.....tri_out = 0;
end

end

%*****
% function write output in .nc *****
%*****
function a_out = write_nc_test(filename,lon,lat,nmonth,m lon_in,m lat_in,...
.....emu_grid_tot)

%% -----
nt=nmonth;
nx=lon;
ny=lat;

m lon = m lon_in;
m lat = m lat_in;
emu_in = emu_grid_tot;

disp('Create file Netcdf...')
filename_in = filename;
filenc = sprintf('./%s.nc', filename_in);
ncid = netcdf.create(filenc,'NC_WRITE');

% Create dimensiones
dimid_lon = netcdf.defDim(ncid,'longitude',nx);
dimid_lat = netcdf.defDim(ncid,'latitude',ny);
%dimid_time = netcdf.defDim(ncid,'time',nt);
dimid_time = netcdf.defDim(ncid,'time',netcdf.getConstant('NC_UNLIMITED'));

% Crear variables y atributos
varid_lon = netcdf.defVar(ncid,'longitude','double',dimid_lon);
netcdf.putAtt(ncid,varid_lon,'long_name','Longitude')
netcdf.putAtt(ncid,varid_lon,'units','degrees_east')
%
varid_lat = netcdf.defVar(ncid,'latitude','double',dimid_lat);
netcdf.putAtt(ncid,varid_lat,'long_name','Latitude')
netcdf.putAtt(ncid,varid_lat,'units','degrees_north')
%
varid_time = netcdf.defVar(ncid,'time','double',dimid_time);
netcdf.putAtt(ncid,varid_time,'long_name','Time')
netcdf.putAtt(ncid,varid_time,'units','Months since 2000-01-01 00:00:00')
%netcdf.putAtt(ncid,varid_time,'units','Months')
%
varid_emu = netcdf.defVar(ncid,'emu_tot','double',[dimid_lon,dimid_lat,dimid_time]);
netcdf.putAtt(ncid,varid_emu,'long_name','Variable')
netcdf.putAtt(ncid,varid_emu,'units','gC/m2')
netcdf.putAtt(ncid,varid_emu,'missing_value',-1e+36)
netcdf.putAtt(ncid,varid_emu,'_FillValue',-1e+36)

netcdf.endDef(ncid)
netcdf.putVar(ncid,varid_lon,m lon);
netcdf.putVar(ncid,varid_lat,m lat);

```

```
%netcdf.putVar(ncid,varid_time,time);  
...  
netcdf.putVar(ncid,varid_emu,[0 0 0],[nx ny nt],emu_in);  
  
netcdf.close(ncid);  
a_out=0;  
%end
```