Vegetation community patterns,

Michel Ferré[1], Induja Pavithran[1] and Hannes Uecker[2],

[1] BGU, michel.ferre.diaz@gmail.com and indujap2013@gmail.com, [2] Institut für Mathematik, Universität Oldenburg hannes.uecker@uol.de,

Abstract

This document serves as supplementary material to the manuscript entitled "Vegetation pattern formation and community assembly under drying climate trends". It presents the `pde2path` implementation of the vegetation community pattern formation model discussed in [FPBUM24]. The spatial domain is discretized using the finite element method, while the trait space is treated via a finite difference approach. We also describe the implementation of the single-species case, including the computation of the Busse Balloon through continuation from bifurcation points.

Contents

1 Introduction. **1**

2 Single species model **3**
2.1 Basic branch continuation 3
2.2 Busse-balloon boundary by branch point continuation 10

3 Community model: `pde2path` implementation. **13**
3.1 Basic BD 18
3.2 Brute force Busse-balloon 20
3.3 Direct numerical simulation: decreasing precipitation 22

1 Introduction.

In [FPBUM24], we study a spatially explicit community model describing the evolution and coexistence of plant species competing for water and light in arid and semi-arid ecosystems. This work is motivated by the threat that climate-driven drying trends pose to ecosystem functioning and the vital services ecosystems provide to humans. To understand the potential responses of plant communities to increasing water stress, the manuscript investigates a mathematical model capturing two key adaptation strategies: (i) community-level shifts from fast-growing to stress-tolerant species, and (ii) spatial self-organization into periodic vegetation patterns.

The community is partitioned into N functional groups, each characterized by a dimensionless trait value $\chi_i \in [0,1]$, with $\chi_1 = 0 < \cdots < \chi_i < \cdots < \chi_N = 1$, representing a trade-off between water stress tolerance ($\chi = 0$) and rapid growth potential ($\chi = 1$). The spatiotemporal dynamics of the vegetation are captured by N biomass density fields $B_i = B_i(\chi_i, x, t)$, corresponding to each functional group. The environmental context is described by two water-related fields: the below-ground water $W = W(x,t)$, which mediates root uptake, and the above-ground water $H = H(x,t)$, associated with surface water dynamics.

A central finding of the study is that, under spatially uniform conditions, the community composition shifts toward more stress-tolerant species as precipitation declines. However, when spatial patterns emerge via a Turing instability, this trend may reverse, favoring fast-growing species again. The spatial plasticity of patterns mediates this reversal, manifested

through *patch thinning* along solution branches and *patch dilution* during wavelength transitions, which buffers further community shifts despite ongoing drying. Bifurcation diagrams and Busse balloon representations of the community dynamics reveal how the initial pattern wavelength and the rate of environmental change influence these buffering effects. The ecological implications of these dynamics are discussed in the context of dryland pasture management and sustainable crop production.

The full model reads: eq:Comm

$$\partial_t B_i = \Lambda W B_i - M_i B_i + D_B \partial_x^2 B_i + D_\chi \partial_\chi^2 B_i, \quad \text{eq:Comm1} \tag{1a}$$

$$\partial_t W = IH - LW - \Gamma W \bar{B} + D_W \partial_x^2 W, \quad \text{eq:Comm2} \tag{1b}$$

$$\partial_t H = P - IH + D_H \partial_x^2 H, \quad \text{eq:Comm3} \tag{1c}$$

where the growth rate of the ith functional group, Λ_i, the infiltration rate of above-ground water into soil, I, and the evaporation rate, L, are written as,

$$\Lambda_i \equiv \frac{\Lambda_0 K_i}{\bar{B} + K_i}, \quad I \equiv \frac{A\left(\bar{\bar{B}} + fQ\right)}{\bar{\bar{B}} + Q}, \quad L \equiv \frac{L_0}{1 + R\bar{B}} \tag{2} \quad \text{eq:TraitQuan}$$

being $\bar{B} \equiv \sum_{i=1}^{N} B_i$ and $\bar{\bar{B}} \equiv \sum_{i=1}^{N} Y_i B_i$; terms that model the interaction of the vegetation community and the above/below ground water content. $\partial_\chi^2 B \equiv N^2 \left(B_{i+1} - 2B_i + B_{i-1}\right)$, with D_χ corresponds to the mutation rate. The trait-dependent terms, $M_i := M_{max} + \chi_i(M_{max} - M_{min})$, $K_i := K_{max} + \chi_i(K_{max} - K_{min})$, and $Y_i := Y_{min} + \chi_i(Y_{max} - Y_{min})$ corresponds to the light capture capacity, mortality and relative contribution to infiltration for each $i-$species group. Notice that the main feedbacks within the vegetation species are given by the total vegetation $\bar{B}(x,t)$ and the relative infiltration contribution $\bar{\bar{B}}(x,t)$, which introduce further complexity into the description. Details are referred to [?]. Here we focus on the implementation of model (1) into the continuation and bifurcation MATLAB package `pde2path` [Uec21, Uec23a].

FPBUM24 eq:Comm necker2021numerical,p2phome

Remark1. a) A technically interesting feature of (??) it the following: Instead of discretizing in χ from the start we could take $\chi \in [0,1]$ as a second "spatial coordinate", and let, for example, $\bar{B}(x,t) = \int B(x,\chi,t)d\chi$, yielding a nonlocal strongly anisotropic 2D problem for $B(x,\chi,t)$ with $\chi-$dependent coefficient in (??). But then $W(x,t)$ and $H(x,t)$ are still only dependent on x and t, and mainly for this reason we found it cleaner to directly start with (??). This aligns with the nature of the theoretical formulation of the problem, trait in vegetation do not corresponds to continuous features among species, they represent a discrete set of characteristics linked to each species and their interaction within landscapes.

eq: Comm eq: Comm1 eq: Comm

b) Additionally, the figures in [?] contain much information, depicting many branches at once, and partly are generated via `pyplot`. For this reason, we provide the basic scripts to produce the mentioned results. Notice that not all states are accessible through branch switching; in some cases, like long-wavelength patterns, direct numerical simulations are needed. The scripts to perform temporal integration of each model, the community model, and the single-species model are provided.

FPBUM24

As follows, the implementation in the MATLAB package `pde2path` of the single-species model and the community model is given.

2 Single species model

For illustrative purposes, we first consider the single-species model, where $i \in \{1, \dots, N\}$. The model reads as

$$\begin{aligned} \partial_t B_i &= \Lambda_i W B_i - M_i B_i + l^2 D_B \partial_x^2 B_i, && \text{(3a)} \\ \partial_t W &= IH - LW - \Gamma W \bar{B} + l^2 D_W \partial_x^2 W, && \text{(3b)} \\ \partial_t H &= P - IH + l^2 D_H \partial_x^2 H && \text{(3c)} \end{aligned}$$

where now, e.g., $\bar{B} \to B_i$, and $\bar{\bar{B}} \to Y_i B_i$ into the trait-dependent terms (2). Notice that we added a scaling parameter, l, to be used later to compute the boundary of Busse balloons (BBs) by branch point continuation (BPC). The model (3) corresponds to a semi-parabolic partial differential equation of reaction-diffusion type. Models of a similar kind are treated in references [Uec21]. Demos available are given in [Uec23a]. The `single` folder comes with the file needed to calculate a minimal version of the bifurcation diagram shown in Figure 1 at reference [?], and those necessary to calculate BB in Fig. 4, same reference. Table ??

Table 1: Scripts and functions in `single`

file	purpose, remarks
`cmds1`	basic branch continuation.
`cmds2`	basic BB boundary computation.
`bwhinit`	initialization of problem struct `p` with standard parameter values, call of `stanpdeo1D` to generate a 1D PDE object (interval, with mesh), initialization of u with u^*, call of `oosetfemops` to generate the FEM matrices, and finally resetting of some `pde2path` parameters to problem-adapted values.
`oosetfemops`	assemble and store the mass matrix M, and the (1-component) Neumann-Laplacian K.
`nodalf`	"nonlinearity", i.e., terms without spatial derivatives, called in `hotintxs`.
`sG,sGjac`	rhs of (??), and Jacobian; these here have a simple standard structure.
`bpjac`	implements $\partial_u(G_u^T \psi)$ for BPC, see [Uec21, §3.6.1].
`sgbra`	mod of library function `stanbra`;

2.1 Basic branch continuation

The function `bwhinit.m` initializes the problem setup by taking as input the domain size `lx`, the number of discretization points `nx`, the model parameters `par`, an initial homogeneous solution state `[b0, w0, h0]`, and the output folder name. On line 12, the command `p = stanparam(p)` generates a `pde2path` problem structure and populates it with standard values (we encourage consulting `stanparam.m` for an overview). Several of these default values are subsequently overwritten to configure the specific problem setup. On line 15, the branch output is redirected to the custom function `sgbra`, a slight modification of the standard output function `stanbra` (see Listing 5). Line 17 creates a standard 1D PDE object, which includes the finite element mesh and associated routines. Notice that the initial solution vector is then constructed by concatenating the field values and parameters: `p.u = [b0; w0; h0; par]`.

```
1 function p = bwhinit(lx, nx, par, b0, w0, h0, dir)
2 % bwhinit.m     Initialization function for the single-species BWH model
3 % Inputs:
4 %   lx  - Length of the spatial domain
5 %   nx  - Number of discretization points in space
```

```
%   par - Vector of model parameters (appended to state vector)
%   b0, w0, h0 - Initial homogeneous values for biomass, soil water, and surface water
%   dir - Directory name for output and problem labeling

%--- Initialize p-structure and set basic fields ---%
p = [];
p = stanparam(p);             % Create and fill base p-structure with default settings
p = setfn(p, dir);            % Set working directory
%--- Assign function handles ---%
p.fuha.outfu = @sgbra;        % Output function for bifurcation diagrams (e.g., biomass averages)
%--- Set up 1D finite element problem ---%
pde = stanpdeo1Db(0, lx, lx/nx);  % Generate standard 1D PDE object over [0, lx] with spacing dx
p.pdeo = pde;
p.vol  = lx;
p.np   = pde.grid.nPoints;
%--- Define system size and degrees of freedom ---%
p.ndim   = 1;         % 1D spatial domain
p.nc.neq = 3;         % Three PDE components: B, W, H
p.nu     = p.np * p.nc.neq;   % Total number of unknowns
p.sol.xi = 0.1 / p.nu;        % Initial arc-length predictor step size
%--- Set initial state ---%
b = b0 * ones(p.np, 1);w = w0 * ones(p.np, 1);h = h0 * ones(p.np, 1);
p.u = [b; w; h; par];         % Concatenate fields with parameter vector
%--- FEM operators and numerical settings ---%
p.sw.sfem = -1;               % Use OOPDE finite element interface
p = oosetfemops(p);           % Generate finite element matrices
%--- Continuation and bifurcation detection settings ---%
p.sw.bifcheck = 2;            % Enable bifurcation detection (secant method)
p.sw.jac      = 1;            % Use numerical Jacobian
p.nc.ilam     = 1;            % Continuation in first parameter (par(1))
%--- Arc-length continuation parameters ---%
p.sol.ds    = 0.01;           % Initial step size
p.nc.dsmin  = 0.01;           % Minimum step size
p.nc.dsmax  = 3;              % Maximum step size
%--- Bifurcation continuation setting ---%
p.sw.qjac = 0;                % Use numerical Jacobian for continuation from bifurcation points
end
```

Listing 1: `cm2D/bwhsingle/bwhinit.m` Minimal initialization commands.

The function `oosetfemops.m` generates the finite element (FEM) matrices required for subsequent computations. Specifically, it constructs the one-component stiffness matrix, stored in `p.mat.K`, which corresponds to the Neumann Laplacian and is used in assembling the system's diffusion terms (e.g., in `sG.m`). Additionally, it generates the mass matrix, stored in `p.mat.M`, which is required in its full system form—not only for time evolution and residual assembly, but also for spectral computations such as eigenvalue analysis.

```
function p=oosetfemops(p) % set FEM operators, homog. Neuman BC by default
[K,M,~]=p.pdeo.fem.assema(p.pdeo.grid,1,1,1); % FEM matrices
p.mat.K=K; p.mat.M=kron(diag([1,1,1]),M); % scalar Lapl., full M
```

Listing 2: `cm2D/bwhsingle/oosetfemops.m` collect the FEM matrices.

Once the system parameters and main pde2path structure are set up, we can implement them to calculate the rhs of our model (3) based on the script sG.m

```
function r = sG(p, u)
% sG.m     Residual function for the single-species BWH model
% Computes the right-hand side of the PDE system for a given state u.
%
% Inputs:
%   p - pde2path problem structure
%   u - state vector [b; w; h; par]
%
% Output:
%   r - residual vector [r1; r2; r3]
%--- Unpack fields and discretization ---%
n = p.np;                          % Number of spatial points
b = u(1:n);                        % Biomass
w = u(n+1:2*n);                    % Soil water
h = u(2*n+1:3*n);                  % Surface water
%--- Unpack parameters ---%
par     = u(p.nu+1:end);          % Parameter vector
pp      = par(1);                  % Precipitation
Lam0    = par(2); Ga     = par(3); % Growth and mortality rates
A       = par(4); R      = par(5); % Infiltration and evaporation
   parameters
L0      = par(6); f      = par(7);
Q       = par(8); Kmin   = par(9); Kmax = par(10);
Mmin    = par(11); Mmax  = par(12);
Ymin    = par(13); Ymax  = par(14);
Db      = par(15); Dw    = par(16); Dh   = par(17); % Diffusion
   coefficients
chi     = par(18); l     = par(19);      % Trait value and domain scaling
%--- FEM matrices ---%
K  = p.mat.K;                     % Stiffness matrix
M  = p.mat.M(1:n,1:n);            % Mass matrix
ov = ones(n,1);                   % Vector of ones (for convenience)
%--- Trait-dependent parameters ---%
Yi  = Ymax + chi * (Ymin - Ymax);     % Infiltration capacity
Mi  = Mmax + chi * (Mmin - Mmax);     % Mortality
Ki  = Kmax + chi * (Kmin - Kmax);     % Growth saturation constant
%--- Model terms ---%
Lam = Lam0 * Ki ./ (b + Ki);              % Growth rate
I   = A * (Yi * b + f * Q) ./ (Yi * b + Q); % Infiltration rate
L   = L0 ./ (1 + R * b);                  % Evaporation rate
%--- PDE residuals ---%
r1 = -M * (Lam .* w .* b - Mi * b)      + l^2 * Db * K * b;
r2 = -M * (I .* h - L .* w - Ga * w .* b) + l^2 * Dw * K * w;
r3 = -M * (pp - I .* h)                   + l^2 * Dh * K * h;
%--- Concatenate residuals ---%
r = [r1; r2; r3];
end
```

Listing 3: c2Dm/bwhsingle/sG.m, implementing the (negative) rhs of model (3)

That's all we need if we use numerical Jacobians by setting up p.sw.jac=0; however, it is highly recommended to implement the Jacobian through a provided function by the user, as well as for precision and stability of the calculations.[1] We provide the Jacobian as follows,

[1]The numerical Jacobian is provided by a finite difference approximation using sG.m.

```
function Gu = sGjac(p, u)
% sGjac.m     Jacobian of the residual function for the single-species BWH model
% Computes the Jacobian matrix Gu = G_u(p, u) used for Newton steps
%
% Inputs:
%   p - pde2path problem structure
%   u - current solution vector [b; w; h; par]
%
% Output:
%   Gu - Jacobian matrix
%--- Unpack state variables ---%
n  = p.np;                          % Number of spatial points
b  = u(1:n);                        % Biomass
w  = u(n+1:2*n);                    % Soil water
h  = u(2*n+1:3*n);                  % Surface water
par = u(p.nu+1:end);                % Parameter vector
%--- Unpack parameters ---%
pp   = par(1);  Lam0 = par(2);  Ga   = par(3);
A    = par(4);  R    = par(5);  L0   = par(6);
f    = par(7);  Q    = par(8);
Kmin = par(9);  Kmax = par(10);
Mmin = par(11); Mmax = par(12);
Ymin = par(13); Ymax = par(14);
Db   = par(15); Dw   = par(16); Dh = par(17);
chi  = par(18);
l    = par(19);                     % Spatial scaling factor
%--- FEM operators and utilities ---%
M  = p.mat.M(1:n, 1:n);             % Mass matrix
K  = p.mat.K;                       % Stiffness matrix
ov = ones(n, 1);                    % Vector of ones
%--- Trait-dependent parameters ---%
Yi = Ymax + chi * (Ymin - Ymax);
Ki = Kmax + chi * (Kmin - Kmax);
Mi = Mmax + chi * (Mmin - Mmax);
%--- Functional responses and derivatives ---%
Lam  = Lam0 * Ki ./ (b + Ki);                          % Growth rate
dLam = -Lam0 * Ki ./ (b + Ki).^2;                      % dLam/db
I    = A * (Yi * b + f * Q) ./ (Yi * b + Q);           % Infiltration
dI   = A * Yi ./ (Yi * b + Q) ...
     - A * Yi * (Yi * b + f * Q) ./ (Yi * b + Q).^2;   % dI/db
L    = L0 ./ (1 + R * b);                              % Evaporation
dL   = -L0 * R ./ (1 + R * b).^2;                      % dL/db
%--- Local Jacobian block entries ---%
f1b = dLam .* w .* b + Lam .* w - Mi;   f1w = Lam .* b;   f1h = zeros(n, 1);
f2b = dI .* h - Ga * w - dL .* w;       f2w = -L - Ga * b; f2h = I;
f3b = -dI .* h;                         f3w = zeros(n, 1); f3h = -I;
%--- Assemble local Jacobian matrix (blockwise, sparse) ---%
Fu = [
    spdiags(f1b, 0, n, n), spdiags(f1w, 0, n, n), spdiags(f1h, 0, n, n);
    spdiags(f2b, 0, n, n), spdiags(f2w, 0, n, n), spdiags(f2h, 0, n, n);
    spdiags(f3b, 0, n, n), spdiags(f3w, 0, n, n), spdiags(f3h, 0, n, n)
];
%--- Combine diffusion and reaction terms ---%
Diffusion = kron( ...
    [l^2 * Db, 0, 0;
```

```
     0, l^2 * Dw, 0;
     0, 0, l^2 * Dh], K);
Gu = Diffusion - M * Fu;
end
```

Listing 4: `c2Dm/bwhsingle/sGjac.m`, implementing $\partial_U G$, i.e., the Jacobian of G.

Finally, we provide the main script `cmds1.m` that calculates some steps in the homogeneous steady state for stress-tolerant species ($\chi = 1$), encounters a Turing bifurcation point (`bpt1` in pde2path language), and subsequently performs a branch switching to obtain the first and second pattern branches.

```
% cmds1.m     Continuation of homogeneous and patterned solutions
% This script computes the homogeneous solution branch and two patterned
   branches (T1, T2)
% for a 1D spatial vegetation model using pde2path. Assumes bwhinit.m
   defines the PDE.
% === Trait-based parameter: chi ===
% The parameter 'chi' represents a phenotypic trait linked to plant
   functionality,
% such as water-use efficiency or rooting strategy. In the supplementary
   material,
% we explore two cases:
%   - chi = 0: corresponds to fast-growing species
%   - chi = 1: corresponds to stress-tolerant species
% These cases help illustrate how different species or functional types can
    affect
% pattern formation in the vegetation model.
close all; keep p2phome;
global p2pglob; p2pglob.ps = 1; % Set plot style (1 = classic, 2 = fancy)
%% === Parameters ===
% Domain and discretization
pp = 290;          % Precipitation (bifurcation parameter)
lx = 85;           % Length of the spatial domain
nx = 300;          % Number of finite element nodes in 1D
% Model parameters
Lam0 = 8; Ga = 10; A = 3000; L0 = 200; f = 0.01;  % Growth, mortality,
   infiltration, etc.
Q = 12; R = 0.7; chimin = 0.0; chimax = 1.0;
Kmin = 6.7; Kmax = 28.93;Mmin = 14.15; Mmax = 20.585;Ymin = 0.069; Ymax =
   0.1041;
% Diffusion coefficients
Db = 1; Dw = 80; Dh = 1800; Dchi = 1e-4;
% Trait and rescaling
chi = 1; l = 1;  % chi = fixed trait value; l = rescaling parameter
Yi = Ymin + chi * (Ymax - Ymin);  % Chi-dependent root-to-shoot ratio
% Parameter vector passed into model definition
par = [pp; Lam0; Ga; A; R; L0; f; Q; Kmin; Kmax; Mmin; Mmax; ...
       Ymin; Ymax; Db; Dw; Dh; chi; l];
% Initial guess for homogeneous steady state (b, w, h)
b0 = 5.25; w0 = 3.15;
h0 = pp * (Yi * b0 + Q) / (A * (Yi * b0 + f * Q));  % Water balance formula
% Output folder
dir0 = 's1'; % usually linked with chi=1
dir = [dir0 '/hom'];
%% === Initialize p-structure and run homogeneous branch ===
p = bwhinit(lx, nx, par, b0, w0, h0, dir);
p.plot.bpcmp = 1;     % Component index to plot during continuation (1 = b)
```

```
p.nc.eigref = -0.3;   % Reference value to calcualte eigenvalues
p.nc.neig = 3;        % Number of eigenvalues to compute along branch
p.sol.ds = -0.1;      % Initial continuation step size (negative = backward
    in parameter)
p.nc.ilam = 1;        % Index of parameter to continue (1 = pp)
p = cont(p, 60);      % Continue for 60 steps along homogeneous branch
%% === Branch switching from Turing bifurcation ===
aux = []; aux.m = 3; aux.besw = 0;
p0 = cswibra(dir, 'bpt1', aux); % Switch at bifurcation point 'bpt1' from /
   hom branch
p0.sw.bifcheck = 2;     % Switch-on bifurcation check and branch switching
p0.nc.neig = 5;         % Number of eigenvalues to compute (more for
   patterned branches)
p0.nc.eigref = -3;      % Reference value to calcualte eigenvalues
p0.sw.spcalc = 1;       % Compute spectrum each step
p0.nc.dsmin = 1e-6;     % Minimum continuation step size
p0.nc.dsmax = 10;       % Maximum continuation step size
p0.nc.tol = 1e-10;      % Tolerance for Newton's method during continuation
%% === Continue patterned branches (T1 and T2) ===
continue_branch(p0,[1],[dir0 '/T1'],1e-3,200);  % T1 branch: 1st Turing
   mode
continue_branch(p0,[0, 1],[dir0 '/T2'],1e-3,200);  % T2 branch: 2nd Turing
   mode
%% === Plot all branches with different green shades ===
% === Colors for each branch ===
colHom = [0.39216      0.83137      0.07451];
colT1 = [0.40784      0.52941      0.25098];
colT2 = [0.64314      0.81176      0.42745];
figure;
hold on;box on
% Plot branches
plotbra([dir0 '/hom'],'cl',colHom,'tyun','--','cmp',1);
plotbra([dir0 '/T1'], 'cl',colT1, 'tyun','--','cmp',1);
plotbra([dir0 '/T2'], 'cl',colT2, 'tyun','--','cmp',1);

xlabel('Precipitation'); ylabel('||u||');
% Example positions
text(270, 4.9,'Hom','Color',colHom,'FontSize',11,'FontWeight','bold');
text(295, 5.2,'Turing 1','Color',colT1,'FontSize',11,'FontWeight','bold');
text(310, 4.7,'Turing 2','Color',colT2,'FontSize',11,'FontWeight','bold');
%% === Helper Function ===
function continue_branch(p0, tauvec, dirname, ds, nsteps)
    % Helper to switch to patterned branch and run continuation
    % Inputs:
    %   p0     = p-structure from cswibra (at bifurcation point)
    %   tauvec = mode selector for gentau (e.g. [1], [0 1], etc.)
    %   dirname= output folder
    %   ds     = step size
    %   nsteps = number of continuation steps
    p = gentau(p0, tauvec);     % Create new branch with transverse mode(s)
    p = setfn(p, dirname);      % Set filename prefix
    p.sol.ds = ds;              % Set step size
    cont(p, nsteps);            % Continue
end
```

Listing 5: `c2Dm/bwhsingle/cmds1.m`

Now we have all set up to compute steady-state solutions branches from the model (3) by

eq: SSM

running `cmds1.m`, yielding the following –shortened– output,

```
    >> cmds1
Problem directory name: s1/hom
step   lambda      y-axis  residual  iter meth   ds       #-EV b(0)
   0  300.00000    5.23399 7.35e-09    2  nat  0.000e+00   0   49.65399
   1  299.90000    5.23124 8.80e-12    2  nat   -0.10000   0   49.62790
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
  59  270.89969    4.34205 6.44e-12    2  nat   -0.50000   0   41.19233
  60  270.39968    4.32456 7.37e-12    2  nat   -0.50000   0   41.02642
   1 possible bifurcation between 270.4 and 269.9, om=0
 mu_r=-8.52986e-07, mu_i=0
<phi,psi>=6.71159e-10,BP
  61  2.70372e+02 (BP, saved to s1/hom/bpt1.mat) bisection steps 10, last ds 0.00024414
  62  269.89968    4.30697 7.80e-12    2  nat   -0.50000   3   40.85954
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
  79  260.89966    3.96985 7.42e-12    2  nat   -1.00000   3   37.66128
  80  259.89966    3.92952 8.41e-12    2  nat   -1.00000   3   37.27868
Timing: total=5.30086, av.step=0.0302019, av.Newton=0.00210061, av.spcalc=0.00616449
```

First Turing branch

```
    lam=270.3719; smallest eigenvalues: -8.53e-07    0.0161    0.0251
using m=3
Problem directory name: s1/T1
creating directory s1/T1
step   lambda      y-axis  residual  iter meth   ds       #-EV b(0)
   1 possible bifurcation between 270.372 and 271.193, om=0
 mu_r=3.70903e-05, mu_i=0
<phi,psi>=-3.14952e-10,BP
   1  2.70495e+02 (BP, saved to s1/T1/bpt1.mat) bisection steps 10, last ds 2.44141e-05
   2  271.19298    4.34117 7.47e-12    3  arc    0.05000   3   41.25940
   3  271.24295    4.34223 8.68e-12    2  nat    0.05000   3   41.27410
   4  271.34291    4.34436 7.08e-12    2  nat    0.10000   3   41.30347
   1 possible bifurcation between 271.343 and 271.543, om=0
 mu_r=-2.80281e-05, mu_i=0
<phi,psi>=-3.26133e-12,BP
   5  2.71347e+02 (BP, saved to s1/T1/bpt2.mat) bisection steps 10, last ds -9.76563e-0
   also saved to s1/T1/pt5.mat
   6  271.54284    4.34862 6.09e-11    2  nat    0.20000   4   41.36214
   7  271.74277    4.35288 1.86e-11    2  nat    0.20000   4   41.42072
   1 possible bifurcation between 271.743 and 272.143, om=0
 mu_r=5.64688e-05, mu_i=0
<phi,psi>=1.48108e-07,BP
   8  2.71887e+02 (BP, saved to s1/T1/bpt3.mat) bisection steps 10, last ds 0.000195313
   9  272.14267    4.36136 1.20e-09    2  nat    0.40000   5   41.53761
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 106  341.34391    5.46895 1.10e-11    3  nat    0.02500   5   58.73031
   1 possible bifurcation between 341.344 and 341.294, om=0
```

```
 mu_r=1.05115e-05, mu_i=0
<phi,psi>=4.9079e-10,BP
 107  3.41316e+02 (BP, saved to s1/T1/bpt4.mat) bisection steps 10, last ds -2.44141e-0
 108  341.29403    5.46407 1.13e-11    3  nat    0.05000   0   58.73642
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 150  310.14468    4.58636 1.20e-11    2  nat    0.80000   0   53.91918
Timing: total=8.83578, av.step=0.0318982, av.Newton=0.00333239, av.spcalc=0.00495385
```

Second Turing branch

```
    Problem directory name: s1/T2
creating directory s1/T2
step   lambda      y-axis  residual  iter meth   ds       #-EV b(0)
   1 possible bifurcation between 270.372 and 270.903, om=0
mu_r=-0.0123881, mu_i=0, no convergence
   1  270.90260    4.33107 1.10e-09    2  arc    0.05000   3   41.16361
   1 possible bifurcation between 270.903 and 270.953, om=0
mu_r=0.0280418, mu_i=0, no convergence
   2  270.95257    4.33207 7.47e-12    2  nat    0.05000   4   41.17817
   3  271.05252    4.33406 7.59e-12    2  nat    0.10000   4   41.20726
   4  271.25243    4.33805 1.19e-10    2  nat    0.20000   4   41.26538
   1 possible bifurcation between 271.252 and 271.652, om=0
 mu_r=-6.60466e-05, mu_i=0
<phi,psi>=4.38396e-07,BP
   5  2.71272e+02 (BP, saved to s1/T2/bpt1.mat) bisection steps 10, last ds -0.00019531
   also saved to s1/T2/pt5.mat
   6  271.65228    4.34602 5.34e-09    2  nat    0.40000   5   41.38138
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 110  338.57778    5.34937 1.33e-11    3  nat    0.10000   5   58.10432
   1 possible bifurcation between 338.578 and 338.378, om=0
 mu_r=-5.69923e-06, mu_i=0
<phi,psi>=-1.26009e-08,BP
 111  3.38525e+02 (BP, saved to s1/T2/bpt2.mat) bisection steps 10, last ds 9.76563e-05
 112  338.37789    5.33730 1.11e-11    3  nat    0.20000   0   58.09821
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 149  309.77825    4.53834 1.08e-11    2  nat    0.80000   0   53.60794
 150  308.97824    4.51805 1.08e-11    2  nat    0.80000   0   53.46958
Timing: total=9.31093, av.step=0.0361994, av.Newton=0.00311805, av.spcalc=0.00558257
```

Notice this is the only output example in this current supplementary material. We plot the results in Figure (1). fig:min_bif_ss

2.2 Busse-balloon boundary by branch point continuation

In numerical bifurcation analysis, special points along solution branches—such as fold points (FPs), branch (bifurcation) points (BPs), and Hopf bifurcation points (HPs)—play a central role in understanding the structure and stability of solutions. These points are associated with critical changes in the spectral properties of the linearized operators, signaling a stability change (FPs), potential symmetry breaking (BPs), or oscillatory instabilities (HPs).

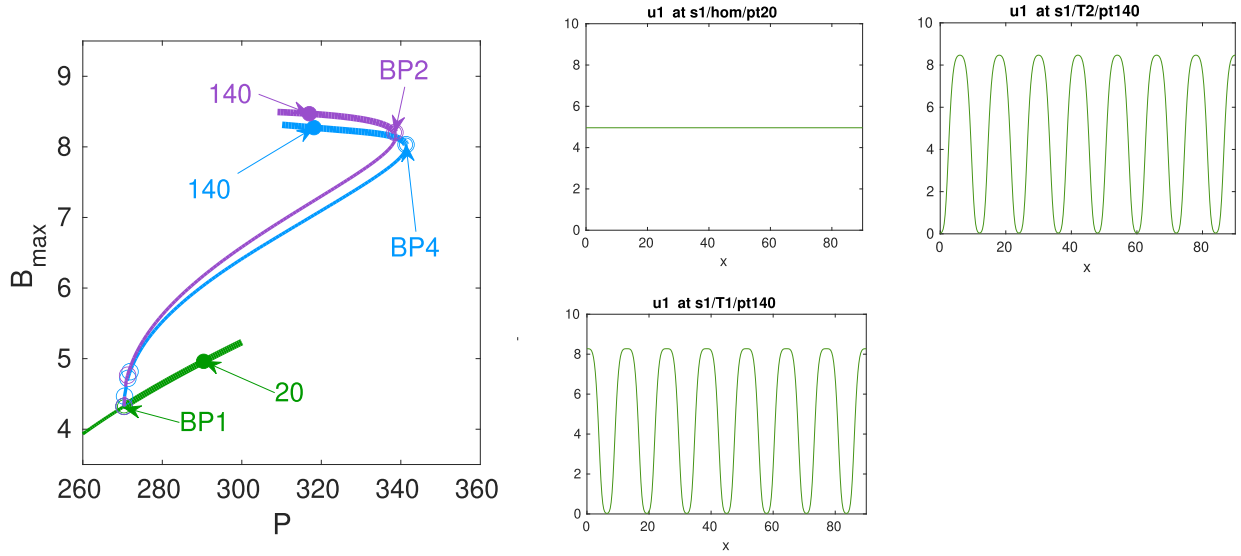

Figure 1: A minimal bifurcation diagram depicting three states by their respective numbers. Bifurcation points are labeled by BP. fig:min_bif_ss

Continuation of such codimension-1 bifurcations with respect to a second system parameter—referred to as FP-continuation (FPC), BP-continuation (BPC), and HP-continuation (HPC)—requires that an additional parameter be treated as free. This allows one to compute curves of bifurcation points in a two-parameter space, which is particularly useful for mapping stability boundaries and organizing bifurcation diagrams. These techniques are standard in numerical bifurcation software (e.g., pde2path in our case) and are discussed in detail in references such as [?, Uec23b]. The numerical continuation of BPs is based on the extended system p2pbook,uecker2023continuation

$$H(U) = \begin{pmatrix} G(u,\lambda) + \mu M_d \psi \\ G_u^T(u,\varphi)\Psi \\ \| \psi \|_2^2 - 1 \\ \langle \psi, G_\lambda(u,\varphi) \rangle \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad U = (u, \psi, \varphi), \tag{4}$$

Here, (u,λ) denotes a simple branch point (BP) for continuation in the primary parameter λ, and ψ is a vector in the kernel of the adjoint linearized operator. The combined parameter vector is written as $\varphi = (\lambda, \mu)$, where $\varphi_1 = \lambda$ is the primary active parameter and $\varphi_2 = \mu$ serves as an additional active parameter. The central task at this stage is to construct $\partial_u(G_u^T\psi)$, which forms the Jacobian of the extended system H; that is,

$$J_H = \begin{pmatrix} G_u & \mu\mathcal{M} & G_\lambda & \mathcal{M}\psi \\ \partial_u(G_u^T\psi) & G_u^T & \partial_\lambda(G_u^T\psi) & 0 \\ 0 & 2\psi^T & 0 & 0 \\ \psi^T\partial_\lambda G_u^T & G_\lambda^T & \psi^T\partial_\lambda G_\lambda & 0 \end{pmatrix} \tag{5}$$

The user-provided derivative matrix $\partial_u(G_u^T\psi)$ is given in file bpjac.m

```
function duGuph=bpjac(p,u) % second derivative for BP continuation
% === Unpack state vector and parameters ===
n = p.np;b = u(1:n);w = u(n+1:2*n);h = u(2*n+1:3*n);
%--- Unpack parameters ---%
pp   = par(1);  Lam0 = par(2);  Ga   = par(3);
```

```
A    = par(4);  R    = par(5);  L0   = par(6);
f    = par(7);  Q    = par(8);
Kmin = par(9);  Kmax = par(10);
Mmin = par(11); Mmax = par(12);
Ymin = par(13); Ymax = par(14);
Db   = par(15); Dw   = par(16); Dh = par(17);
chi  = par(18);
l    = par(19);                  % Spatial scaling factor
% Derived trait-dependent quantities
ov  = ones(n,1);
Yi  = Ymax + chi * (Ymin - Ymax);
Ki  = Kmax + chi * (Kmin - Kmax);
Mi  = Mmax + chi * (Mmin - Mmax);
% === Partial derivatives for each equation ===
% Biomass equation (row 1)
f1bb = -2 * (Lam0 * w * Ki^2) ./ (b + Ki).^3;
f1wb =  (Lam0 * Ki^2) * ov ./ (b + Ki).^2;
% Water equation (row 2)
f2bb = -2 * Lam0 * R^2 * w ./ (1 + R*b).^3 ...
       + 2 * A * Q * Yi^2 * (1 - f) * h ./ (Q + Yi*b).^3;
f2bw = -Ga * ov + L0 * R * ov ./ (1 + R*b).^2;
f2bh = -A * Q * Yi * (1 - f) * ov ./ (Q + Yi*b).^2;
% Surface water equation (row 3)
f3bb = -2 * A * (1 - f) * Q * Yi^2 * h ./ (Q + Yi*b).^3;
f3bh =  A * (1 - f) * Q * Yi * ov ./ (Q + Yi*b).^2;
% === Scalar products with perturbation fields ===
ph1 = u(p.nu + 1         : p.nu +     n);
ph2 = u(p.nu + 1 +   n   : p.nu + 2 * n);
ph3 = u(p.nu + 1 + 2 * n : p.nu + 3 * n);
% === Construct diagonal matrices for Frech t derivatives ===
M1 = spdiags(f1bb .* ph1 + f2bb .* ph2 + f3bb .* ph3, 0, n, n);
M2 = spdiags(f1wb .* ph1 + f2bw .* ph2,               0, n, n);
M3 = spdiags(f2bh .* ph2 + f3bh .* ph3,               0, n, n);
% === Block matrix Fu construction ===
Z = sparse(n,n);  % Zero block for structure
Fu = [ M1  M2  M3;
       M2   Z   Z;
       M3   Z   Z ];
% === Compute matrix-vector product ===
M = p.mat.M;
duGuph = - Fu * M;
```

Listing 6: `cm2D/bwhsingle/bpjac.m` numerical user-provided $\partial_u(G_u^T\psi)$ matrix.

The script `cmds2.m` for BP's continuation reads as follows,

```
% bpcontT1.m     Bifurcation point continuation from subcritical Turing
   bifurcation
% This script tracks the location of a subcritical bifurcation point (bpt6)
% on the T1 branch in a 2D parameter space, forming a Busse balloon domain.
close all; keep p2phome;
%% === Initialization: Bifurcation Point Continuation ===
% Initialize continuation from bifurcation point 'bpt6' on branch s1/T1
% We continue with respect to parameter indices 1 and 19
%  - 1 = precipitation (P)
%  - 19 = l (domain length rescaling or trait scaling)
% Output folder: 'bps1a'
p = bpcontini('s1/T1', 'bpt6', 19, 'bps1a', 3e-3);  % Start from bif point
   with ds = 3e-3
```

```
% Plotting setup
p.plot.bpcmp = 5;      % Plot 5th component during continuation
p.sw.spjac = 1;        % Use sparse Jacobian for efficiency
p.nc.dsmax = 1;        % Max step size
p.nc.dsmin = 1e-5;     % Min step size
p.nc.del   = 4e-3;     % Arclength increment constraint (step size control)
p.nc.lammin = 0;       % Lower bound for continuation parameter
% Jacobian for bifurcation point continuation
p.fuha.spjac = @bpjac;
% Disable spectrum calculation and bifurcation detection
p.sw.spcalc = 0;
p.sw.bifcheck = 0;
% Additional numerical tuning
p.nc.almine = 0.4;      % Minimum arclength weight for continuation
p.nc.tol = 8e-7;        % Newton tolerance
p.file.smod = 10;       % Save every 10th solution point
% Clean up old files if present and start continuation
huclean(p);             % Remove any previous output files in the folder
p = cont(p, 600);       % Continue for 600 steps
%% === Reverse Continuation ===
% Load initial point from the just-computed branch
p = loadp('bps1a', 'pt0');
p.sol.ds = -1.3 * p.sol.ds;  % Reverse direction with slightly larger step
p = setfn(p, 'bps1b');       % Set new output folder for reverse run
p = cont(p, 200);            % Continue for 200 steps in reverse
%% === Load Data and Extract Bifurcation Path ===
% Load branch point continuation results
p0 = loadpp('bps1a');
pp0 = p0.branch(11,:);   % Parameter 1 values (e.g. precipitation P)
l0  = p0.branch(4,:);    % Parameter 2 values (e.g. domain length l)
p1 = loadpp('bps1b'); pp1 = p1.branch(11,:);l1  = p1.branch(4,:);
%% === Plotting the Busse Balloon ===
% Compute wavenumber from domain length:
% Assume 1D domain of length lx = 80 with 6.5 wavelengths     wl = 80 / 6.5
% Then k = 2   / wl = 2   / (80 / 6.5)     0.51051
kfactor = 0.51051;  % Converts domain rescaling l into wavenumber k =
    kfactor * l
figure;
hold on;box on
plot(pp0, kfactor * l0, 'LineWidth', 2, 'Color', 'b');  % Forward BP curve
plot(pp1, kfactor * l1, 'LineWidth', 2, 'Color', 'b');  % Reverse BP curve
% Axes and labels
xlabel('Precipitation');
ylabel('Wavenumber');
title('Busse Balloon/Single Species');
set(gca, 'FontSize', 14);
```

Listing 7: `cm2D/bwhsingle/cmds2.m` numerical computing Busse-balloon boundaries by BPs-continuation.

The results are shown in Figure

3 Community model: `pde2path` implementation.

Our objective here is the implementation of the community model (1) in `pde2path`. As noted in Remark 1(a), this model deviates from *standard* reaction–diffusion systems due to the inclusion

of N biomass fields, $B_1, \ldots, B_N$. This requires several adjustments:

- Additional effort is needed to implement the right-hand side $-G(u)$ of (1) in the function `sG.m`, as well as its Jacobian in `sGjac.m`. However, once `sG.m` is set up, the structure of `sGjac.m` follows in a relatively straightforward manner.
- Minor modifications are required in the plotting function `userplot.m`, where we also wish to display spatial averages $\bar{B}$ during continuation, and plot solutions independently using `plotsol` when `p.plot.pstyle = -1`. To facilitate switching between different plotting styles, we define multiple versions of `userplot`, and use the (user-defined) global variable `p2pglob` to pass a plotting mode switch `p2pglob.ps` to `userplot`, which then calls the appropriate plotting routine (`uplot1`, `uplot2`, or `uplot3`).
- We also apply a minor modification to the library function `tintxs`, which is used for direct numerical simulations (DNS), both for generating initial guesses for continuation and for traversing the Busse balloon.

Once these elements are in place, the continuation process proceeds as usual. The general data layout remains the same: we define the solution vector as $u = [B_1, \ldots, B_N, W, H, \texttt{pars}]^T$, where all biomass and water components are discretized on `np` grid points over a one-dimensional domain of length `lx`. The model parameters from (1) are stored in the vector `par`, as is standard in `pde2path`. Altogether, this results in `p.nu = (N+2)*p.np` degrees of freedom (DoF), arising from the concatenation of the N biomass fields, the water variables, and the system parameters.

Table 2 provides an overview of the fields used in the `pde2path` implementation of the community model.

Table 2: Scripts and functions in `bwhComm`.

file	purpose, remarks
`bwhinit`	initialization of problem struct `p` with standard parameter values, call of `stanpdeo1D` to generate a 1D PDE object (interval, with mesh), initialization of u with u^*, call of `oosetfemops` to generate the FEM matrices, and finally resetting of some `pde2path` parameters to problem-adapted values.
`oosetfemops`	assemble and store the mass matrix M, and the (1-component) Neumann-Laplacian K.
`nodalf`	"nonlinearity", i.e., terms without spatial derivatives.
`sG,sGjac`	rhs of (1), and Jacobian; these here have a simple standard structure.
`sgbra`	output function that provides different functions to be plotted during continuation like χ_{max}.
`userplot`	function that defines the plotting solution windows during continuation
`bbdns`	...
`cmds1`	main script, containing basic `pde2path`commands, to be run cell-by-cell

tb:community

In `oosetfemops` we set up the (one-component Neumann Laplacian) stiffness matrix stored in `p.mat.K`, and the full system mass matrix, stored in `p.mat.M`, given by the diagonal block allocation of one-component mass matrix. These are then used in `sG.m` to assemble the system's diffusion terms and the calculation of the FEM integrals related to the nonlinearities.

For clarity, the rhs $-G(u)$ of the model (1), component wise, is given as follows: eq:Comm

$$\begin{aligned} G_i &:= -\left(\Lambda_i W B_i - M_i B_i + D_\chi N^2 (B_i - 2B_i + B_{i-1}\right) + D_B K B_i, \\ G_{N+1} &:= -\left(IH - LW - \Gamma W \bar{B}\right) + D_W K W, \\ G_{N+2} &:= -\left(P - IH\right) + D_H K H, \end{aligned}$$

where $i = (1, \ldots, N)$. Due to the complications that were mentioned previously, `sG.m` is given explicitly,

sG

```
function r = sG(p, u)
% sG     RHS of the bwh-community model with explicit chi-diffusion
% This version treats chi-diffusion explicitly (e.g., for IMEX time integration),
% and omits spatial diffusion (Db = Dw = Dh = 0).
% --- Unpack model dimensions and state variables ---
N = p.N;                  % Number of traits
n = p.np;                 % Number of spatial grid points
% Reshape biomass field B(x,   ) into a (space    trait) matrix
b = reshape(u(1:N*n), n, N);
% Extract scalar fields (soil water and surface water)
w = u(N*n+1 : (N+1)*n);
h = u((N+1)*n+1 : (N+2)*n);
% --- Extract parameters from the solution vector ---
par = u(p.nu+1:end);
[pp, Lam0, Ga, A, R, L0, f, Q, ...
 Kmin, Kmax, Mmin, Mmax, Ymin, Ymax, ...
 Db, Dw, Dh, Dchi, chimin, chimax] = deal( ...
    par(1), par(2), par(3), par(4), par(5), ...
    par(6), par(7), par(8), par(9), par(10), ...
    par(11), par(12), par(13), par(14), par(15), ...
    par(16), par(17), par(18), par(19), par(20));
% --- Retrieve FEM matrices ---
K = p.mat.K;              % Spatial stiffness (Laplacian) matrix
M = p.mat.M(1:n,1:n);     % Mass matrix for scalar fields
% --- Initialize variables ---
ov   = ones(n,1);         % All-ones vector
bt   = zeros(n,1);        % Total biomass ( b  )
btt  = zeros(n,1);        % Weighted biomass ( b    )
r    = zeros(N*n,1);      % Output for biomass equations
% Trait mesh size
dchi = (chimax - chimin) / (N - 1);
% === Compute total and weighted biomass ===
for i = 1:N
    chii = chimin + (i-1)*dchi;                          % Trait value  _i
    bt   = bt + b(:,i);                                  % Total biomass
    Yi   = Ymax + chii * (Ymin - Ymax);                  % Trait-dependent water-use efficiency
    btt  = btt + b(:,i) * Yi;                            % Weighted biomass
end
% === Infiltration and evaporation functions ===
I = A * (btt + f*Q) ./ (btt + Q);                        % Infiltration rate I(x)
L = L0 ./ (1 + R * bt);                                  % Evaporation rate L(x)
% === Biomass equations for each trait ===
for i = 1:N
```

```
44      chii = chimin + (i-1)*dchi;
45      % Trait-dependent coefficients
46      Ki    = Kmax + chii * (Kmin - Kmax);
47      Mi    = Mmax + chii * (Mmin - Mmax);
48      Lami = Lam0 * Ki ./ (bt + Ki);                          % Growth
        efficiency
49      bi = b(:,i);                                             % Biomass for
        trait i
50      % Trait diffusion term (finite difference in    with Neumann BCs)
51      switch i
52          case 1
53              bcc = (b(:,2) - 2*b(:,1)) / dchi^2;             % Left boundary
54          case N
55              bcc = (-2*b(:,N) + b(:,N-1)) / dchi^2;          % Right boundary
56          otherwise
57              bcc = (b(:,i+1) - 2*b(:,i) + b(:,i-1)) / dchi^2;
58      end
59      % Biomass dynamics equation:
60      % r1 = -M * [ growth - mortality + trait diffusion ] + spatial
        diffusion
61      r1 = -M * (Lami .* w .* bi - Mi * bi + Dchi * bcc) + Db * K * bi;
62      % Store result into appropriate block in r
63      r((i-1)*n+1 : i*n) = r1;
64  end
65  % === Water equation ===
66  % r2 = -M * (I*h - L*w - Ga*bt*w) + Dw*K*w
67  r2 = -M * (I .* h - L .* w - Ga * bt .* w) + Dw * K * w;
68  % === Surface water equation ===
69  % r3 = -M * (pp - I*h) + Dh*K*h
70  r3 = -M * (pp - I .* h) + Dh * K * h;
71  % === Assemble full residual vector ===
72  r = [r; r2; r3];
```

Listing 8: `c2Dm/bwhcom/sG.m` provides the calculation of the complete RHS of the system of equations (11) in finite element formulation. Notice that in line 9, the diffusion coefficients are provided accordingly. eq:Comm

The Jacobian function `sGjac.m` is inside the supplemental material folder. We encourage the reader to revise it.

The initialization file `bwhinit.m` is quite similar to the single species model (3), except that we need to initialize the N fields, in a χ-dependent way, and set `p.plot.pstyle=-1` for our plotting needs[2] eq: SSM

bwhinit

```
1  % bwhinit.m     Initialization function for the BWH community model
2  % This function sets up a pde2path problem structure ‘p‘ for a 1D trait-
       structured
3  %  b i o m a s s watersurface  water model, using the OOPDE finite element
       framework.
4  % It defines spatial and trait discretizations, boundary conditions, and
       initial states.
5  function p = bwhinit(lx, nx, N, par, dir, aux)
6  % === Create base p-structure and assign function handles ===
7  p = [];
8  p = stanparam(p);                      % Initialize standard p-structure
```

[2]Please, compare both initialization files, the single and community cases, in a way to get us to `pde2path` implementations.

```
p = setfn(p, dir);                      % Set directory for results
p.fuha.outfu = @sgbra;                  % Output function for branch data
p.sw.verb = 2;                          % Verbosity level
% === Define PDE domain (1D) and FEM discretization ===
pde = stanpdeo1Db(0, lx, lx / nx);      % Standard 1D pdeo object over [0, lx]
    with nx elements
p.np = pde.grid.nPoints;                % Number of spatial grid points
p.pdeo = pde;
n = p.np;                               % Alias for readability
% === Define system and trait discretization ===
p.N = N;                                % Number of trait discretization bins
p.nc.neq = N + 2;                       % Total number of equations (N traits
   + w + h)
p.ndim = 1;                             % Spatial dimension
p.vol = lx;                             % Domain length
p.nu = p.np * p.nc.neq;                 % Total number of degrees of freedom
p.sol.xi = 0.1 / p.nu;                  % Arclength continuation parameter
% === Trait discretization and initial condition setup ===
ov = ones(n, 1);                        % Vector of ones for spatial profile
b = zeros(n * N, 1);                    % Initialize biomass component
x = getpte(p);                          % Get spatial node positions
chimin = par(19); chimax = par(20);
delchi = (chimax - chimin) / (N - 1);
switch aux.sw
    case 1  % Localized phenotype near chi = 0.825
        for i = 1:N
            chi = chimin + (i - 1) * delchi;
            b((i - 1) * n + 1 : i * n) = 10 * sech(28 * (chi - 0.825)).^2;
        end
        w = 0.9 * ov;         h = 3.5 * ov;
    case 2  % Periodic phenotype structure near chi = 0.36
        for i = 1:N
            chi = chimin + (i - 1) * delchi;
            b((i - 1) * n + 1 : i * n) = 8 * sech(28 * (chi - 0.36)) .* cos
   ((2 * pi * 10.8 / p.vol) * x).^2;
        end
        w = 15 * ov;         h = 28 * ov;
    case 3  % Weak phenotype expression centered at chi = 0.5
        for i = 1:N
            chi = chimin + (i - 1) * delchi;
            b((i - 1) * n + 1 : i * n) = 0.5 * sech(20 * (chi - 0.5)).^2;
        end
        w = 35 * ov;         h = 125 * ov;
end
% === Assemble full solution vector and finalize FEM setup ===
p.u = [b; w; h; par];     % Initial state: biomass (vector), w, h, and
   parameters
p.sw.sfem = -1;           % Use OOPDE FEM engine
p = oosetfemops(p);       % Assemble FEM matrices
% === Plotting and continuation parameters ===
p.plot.pstyle = -1;       % Use user-defined plotting style
p.plot.bpcmp = 2;         % Branch plotting component (e.g. water)
p.plot.pcmp = 2;          % Solution plotting component
p.nc.ilam = 1;            % Index of parameter to continue (1 = pp)
p.frcut=0.0001;
end
```

Listing 9: `cm2D/bwhcom/bwhinit.m` collects the minimal initialization commands needed.

3.1 Basic BD

The script `cmds1.m` computes and plots only two Turing branches from the homogeneous steady state, and subsequently, computes a third branch, which accounts for the snaking branch of localized patterns.

cmds1

```
close all; keep p2phome;
global p2pglob;
p2pglob.gu = [];    % GUI off
p2pglob.nzi = 0;    % sparse Jacobian (no info printing)
p2pglob.ps = 1;     % plot style
%% === Parameters and Initialization ===
% Domain and discretization
lx = 90;            % spatial domain length
nx = 350;           % number of grid points
N  = 35;            % number of trait bins (chi values)
% Initial condition selector
aux.solve = 1;
aux.sw    = 1;       % selects type of initial trait-distribution
% System parameters
pp = 300; Lam0 = 8; Ga = 10; A = 3000; L0 = 200; f = 0.01;
Q = 12; R = 0.7;
chimin = 0; chimax = 1;
Kmin = 6.7; Kmax = 35.599;
Mmin = 14.15; Mmax = 22.515;
Ymin = 0.069; Ymax = 0.11463;
Db = 1; Dw = 80; Dh = 1800; Dchi = 1e-4;
% Parameter vector for the model
par = [pp; Lam0; Ga; A; R; L0; f; Q;
       Kmin; Kmax; Mmin; Mmax; Ymin; Ymax;
       Db; Dw; Dh; Dchi; chimin; chimax];
% Output folders
dir0 = 'comm';
dir  = [dir0 '/hom'];
% Initialize problem structure
p = bwhinit(lx, nx, N, par, dir, aux);
%% === Time Integration to Relax to Steady State ===
% Useful for a good initial guess for Newton
t1 = 0; ts = [];
dt = 3e-3; nt = 8e4; nc = 0;
pmod = nt / 10; smod = pmod;
[p, t1, ts, nc] = tintxs(p, t1, ts, dt, nt, nc, pmod, smod, @sGdns);
%% === Newton Iteration to Find Steady State ===
p.nc.tol = 1e-11;p.nc.almin=0.25;p.nc.bisecmax=15;
[p.u, p.r, iter] = nloop(p, p.u);
fprintf('res = %g, iter = %i\n', norm(p.r, Inf), iter);
plotsol(p);
%% checking jac; Best option is assembled Gu
p.fuha.sGjac=@sGjac; % brute force, slightly faster!
[Gu,Gn]=jaccheck(p);Gd=abs(Gu-Gn); em=max(max(Gd));
mclf(20); spy(Gd>em/2);
%% === Homogeneous Branch Continuation ===
p.sol.ds = -5e-1;          % Initial continuation step size
p.sw.bifcheck = 2;       % Enable bifurcation detection
p.nc.tol = 1e-8;        % Tolerance for Newton step during continuation
p.nc.eigref=-0.1;
p.sw.spcalc=1;
```

```
tic;
p = cont(p, 40);        % Continue for 30 steps
toc;
%% === Branch Switching at Turing Point (bpt1) ===
aux = []; aux.m = 3; aux.besw = 0;
p0 = cswibra(dir, 'bpt1', aux);
% Configure eigenvalue and numerical tolerances
p0.nc.eigref = -3; p0.nc.neig   = 3;
p0.nc.dsmin  = 1e-6; p0.nc.dsmax  = 2;
p0.nc.tol    = 1e-7;
%% === Continue T1 Branch (First Turing Mode) ===
dirT = [dir0 '/T1'];
p = gentau(p0, [1]);     % Projection onto first mode
p = setfn(p, dirT);
p.sol.ds = 1e-3;
p = cont(p, 300);
%% === Continue T2 Branch (Second Turing Mode) ===
dirT = [dir0 '/T2'];
p = gentau(p0, [0 1]);   % Projection onto second mode
p = setfn(p, dirT);
p.sol.ds = 1e-1;
p = cont(p, 300);
%% === Branch Switching to Localized patterns branch  (bpt1) ===
dir  = [dir0 '/T1'];
aux = []; aux.m = 3; aux.besw = 0;
p0 = cswibra(dir, 'bpt2', aux);
% Configure eigenvalue and numerical tolerances
p0.nc.eigref = -3; p0.nc.neig   = 3;
p0.nc.dsmin  = 1e-6; p0.nc.dsmax  = 2;
p0.nc.tol    = 1e-7;

%% === Continue T1 Branch (First Turing Mode) ===
dirT = [dir0 '/T1s'];
p = gentau(p0, [1]);     % Projection onto first mode
p = setfn(p, dirT);
p.sol.ds = 1e-3;
p = cont(p, 900);

%% === Plot all branches with different green shades ===
% === Colors for each branch ===
colHom = [0.39, 0.83, 0.07];
colT1 = [0.40, 0.53, 0.25];
colT2 = [0.64, 0.81, 0.43];
colT1s = [0.13, 0.7, 0.67];

% Homogeneous solution branch
plotbra('comm/hom', 'cl', colHom, ...
        'tyun', '--', 'cmp', 3,'lab',20);
% T1 (first pattern)
plotbra('comm/T1', 'cl', colT1, ...
        'tyun', '--', 'cmp', 3,'lab',340);
% T2 (second pattern)
plotbra('comm/T2', 'cl', colT2, ...
        'tyun', '--', 'cmp', 3,'lab',260);
% T1s (localized pattern)
plotbra('comm/T1s', 'cl', colT1s, ...
        'tyun', '--', 'cmp', 3,'lab',830);
```

```
ylabel('$ \chi_{max}$', 'Interpreter', 'latex');
xlabel('$P$', 'Interpreter', 'latex');
% Example positions (adjust as needed)
text(302, 0.66, 'Hom', 'Color', colHom, 'FontSize', 11, 'FontWeight', 'bold
    ');
text(400, 0.43, 'Turing 1', 'Color', colT1, 'FontSize', 11, 'FontWeight', '
    bold');
text(338, 0.46, 'Turing 2', 'Color', colT2, 'FontSize', 11, 'FontWeight', '
    bold');
text(360, 0.59, 'Localized pattern', 'Color', colT1s, 'FontSize', 11, '
    FontWeight', 'bold');
box on
%% === Plotting Solution Snapshots ===
% These correspond to specific solution points along the branches
plotsol('comm/hom', 'pt20',  'pfig', 1);  % Homogeneous state
plotsol('comm/T1',  'pt340', 'pfig', 2);  % First patterned state
plotsol('comm/T2',  'pt260', 'pfig', 3);  % Second patterned state
plotsol('comm/T1s',  'pt830', 'pfig', 3);  % Localized patter state
```

Listing 10: `cm2D/bwhcom/cmds1.m` Basic continuation comands.

`cmds1` yields the following results, shown in Figure.

3.2 Brute force Busse-balloon

In reference [?], we investigate the evolution of plant communities under varying precipitation levels. We show that, for spatially homogeneous solutions, the community composition shifts toward stress-tolerant species as precipitation decreases. In contrast, when periodic patch states emerge (promoting the increased water availability at each patch surrounding), the composition shifts back toward fast-growing species, exhibiting *patch thinning* along the solution branch as precipitation continues to decline. These dynamics are illustrated in Figure 3 of [?].

To compare the patterned community states with those of single-species systems at the trait extremes ($\chi = 1$ and $\chi = 0$), we characterize and plot the corresponding Busse balloons, as shown in Figure 4 of [?]. This comparison reveals key ecological effects of species coexistence on pattern stability. Specifically, we find that the presence of multiple functional groups promotes pattern stability under lower precipitation levels—extending the stability region downward—while increased community-level water uptake leads to a contraction of the stability region at higher precipitation levels, compared to the single-species cases.

Busse balloon diagrams are typically computed via bifurcation point continuation, which involves doubling the system size to accommodate the extended eigenvalue problem. However, due to technical limitations (e.g., memory constraints), this method was not feasible for the full community model. As a result, we employed a brute-force approach: computing periodic solution branches, extracting their stable portions, and interpolating the boundaries to produce a smoothed Busse balloon.

As a workaround, we employed a brute-force method: we computed a series of periodic solution branches, identified their stable segments, and interpolated the boundaries to construct a smoothed Busse balloon. The code for this brute-force Busse balloon computation is provided below.

cmds1

```
function [kv, pv] = bfbb(kv, pv, dir, k0)
% bfbb     Brute Force Busse Balloon
% Appends stable solutions along a continuation branch to build
% a Busse balloon curve (i.e., (P, k) values where patterns are stable).
% Inputs:
```

```
%   kv   - existing vector of wavenumbers (can be empty [])
%   pv   - existing vector of parameter values (e.g., precipitation P)
%   dir  - folder name of the solution branch (e.g., 'comm/T1')
%   k0   - wavenumber associated with this branch (user-supplied or
   estimated)
% Outputs:
%   kv   - extended wavenumber vector (only stable points included)
%   pv   - extended parameter vector for those stable solutions
% === Load branch data from directory ===
p = loadpp(dir);           % Load full solution structure
pv0 = p.branch(4, :);      % Parameter values (e.g., precipitation)
inv = p.branch(3, :);      % Stability indicator: # of unstable eigenvalues
% === Loop through all solution points on the branch ===
lb = length(pv0);
for i = 1:lb
    if inv(i) < 1        % stable solution: no unstable modes
        kv = [kv, k0];   % append wavenumber
        pv = [pv, pv0(i)]; % append parameter
    end
end
```

Listing 11: `cm2D/bwhcom/bfbb.m` Brute force Bussee-balloon function.

By considering the original data for this purpose, `cmds2.m` obtain the raw results shown in Figure 4 for the community model,

cmds1

```
close all; clc; clear all;
keep p2phome; global p2pglob; p2pglob.gu = [];
p2pglob.nzi = 0; p2pglob.ps = 1;
%% === Brute-force Busse Balloon Construction (chi = 0.88) ===
% This script scans multiple Turing branches and collects stable points
% for plotting the stability region (Busse Balloon)
kvcm0 = [];  % Wavenumber values
pvcm0 = [];  % Precipitation (or control parameter) values
% List of folders and associated wavenumbers
branchList = {
    '11/T1',  0.58643;
    '11/T2',  0.56549;
    '11/T3',  0.54454;
    '11/T4',  0.62832;
    '11/T5',  0.60737;
    '11/T6',  0.5236;
    '11/T7',  0.64926;
    '11/T8',  0.67021;
    '11/T9',  0.73304;
    '11/T10', 0.77493;
    '11/T11', 0.81681;
    '11/T12', 0.87965;
    '11/T13', 0.90059;
    '11/T14', 0.46077;
    '11/T15', 0.43982;
    '11/T16', 0.41888;
    '11/T17', 0.39794;
    '11/T18', 0.79587;
    '11/T19', 0.83776;
    '11/T20', 0.8587;
    '11/T21', 0.75398;
    '11/T22', 0.71209;
```

```
    '11/T23', 0.48171;
    '11/T24', 0.69115;
    '11/T25', 0.50265;
    '11/T26', 0.35605;
    '11/T27', 0.33510;
    '11/T28', 0.31416;
    '11/T29', 0.27227;
    '11/T30', 0.23038;
    '11/T31', 0.18850;
    '11/T32', 0.14661;
    '11/T33', 0.10472;
    '11/T34', 0.083776;
    '11/T35', 0.20944;
    '11/T36', 0.16755;
    '11/T37', 0.12566;
    '11/T38', 0.25133;
    '11/T39', 0.29322;
    '11/T40', 0.37699;
};
% === Loop through all branches and collect stable solutions
for i = 1:size(branchList, 1)
    dir = branchList{i, 1};
    k0  = branchList{i, 2};
    [kvcm0, pvcm0] = bfbb(kvcm0, pvcm0, dir, k0);
end
%% === Plot Busse Balloon
mclf(17);
hold on;
plot(pvcm0, kvcm0, 'x', 'Color', 'm');  % Stable points from brute force
% Optional: overlay bifurcation curves (must be defined elsewhere)
% Example: overlay continuation of bifurcation points (Turing threshold)
% plot(pp0, 0.07854 * l0, 'LineWidth', 2, 'Color', 'b');
% plot(pp1, 0.07854 * l1, 'LineWidth', 2, 'Color', 'r');
% Axes settings
ylim([0, 1.2]);
xlabel('Precipitation $P$', 'Interpreter', 'latex');
ylabel('Wavenumber $k$', 'Interpreter', 'latex');
title('Busse Balloon (Stable Pattern Region)', 'Interpreter', 'latex');
set(gca, 'FontSize', 14);
```

Listing 12: `cm2D/bwhcom/cmds2.m` Brute force Bussee-balloon calculation from the original data.

3.3 Direct numerical simulation: decreasing precipitation

Induja

References

[Uec21] Hannes Uecker. *Numerical continuation and bifurcation in Nonlinear PDEs*. SIAM, 2021. uecker2021numerical p2phome

[Uec23a] H. Uecker. pde2path – a matlab package for continuation and bifurcation in systems of pdes, v3.1, 2023. uecker2023continuation

[Uec23b] Hannes Uecker. Continuation of fold points, branch points, and hopf points with constraints in pde2path. 2023.