# Numerical continuation and simulation of a spatial model for water-limited plant communities

Michel Ferré[1], Induja Pavithran[1], Bidesh K. Bera[1], Hannes Uecker[2], and Ehud Meron[1]

[1] The Swiss Institute for Dryland Environmental and Energy Research, BIDR, Ben-Gurion University of the Negev, Sede Boqer Campus, Israel.

[2] Institut für Mathematik, Universität Oldenburg, Germany.

Corresponding authors: michel.ferre.diaz@gmail.com, indujap2013@gmail.com, hannes.uecker@uol.de

**Abstract**

This document describes the codes for the manuscript entitled "Vegetation pattern formation and community assembly under drying climate trends" by Ferré et al.. The methods include numerical continuation using `pde2path` and direct numerical simulations.

# Contents



# 1 Introduction.

In [FPB$^+$25], we study a spatially explicit model for a community of plants competing for water and light. We use a trait-based approach to describe the community. Accordingly, the community is partitioned into $N$ functional groups, each characterized by the value of a dimensionless trait parameter $\chi \in [0,1]$, with $\chi_1 = 0 < ... < \chi_i < ... < \chi_N = 1$, representing a trade-off between fast growth ($\chi = 0$) and water-stress tolerance ($\chi = 1$). The model then consists of biomass density fields for each functional group, $B_i = B_i(\chi_i, x, t)$, and two fields describing below-ground water $W = W(x,t)$, accessible to the plants' roots, and above-ground water $H = H(x,t)$, associated with surface water dynamics; $t \geq 0$ is time, and $x \in \Omega$ represents space, where for the domain $\Omega \subset \mathbb{R}$ we choose large bounded intervals, and pose homogeneous Neumann boundary conditions (aka no flux BCs) on $\partial\Omega$. The model reads:

$$\partial_t B_i = \Lambda_i W B_i - M_i B_i + D_B \partial_x^2 B_i + D_\chi \partial_\chi^2 B_i, \tag{1a}$$

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

$$\partial_t H = P - IH + D_H \partial_x^2 H, \tag{1c}$$

where the growth rate of the $i$th functional group, $\Lambda_i$, the infiltration rate of above-ground water into soil, $I$, and the evaporation rate, $L$, are given by,

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

with $\bar{B} \equiv \sum_{i=1}^{N} B_i$ and $\bar{\bar{B}} \equiv \sum_{i=1}^{N} Y_i B_i$. In (1a), $\partial_\chi^2 B_i \equiv N^2 (B_{i+1} - 2B_i + B_{i-1})$, where $D_\chi$ represents the mutation rate. The trait-dependent terms,

$$\begin{aligned} M_i &:= M_{max} - \chi_i (M_{max} - M_{min}), && (3)\\ K_i &:= K_{max} - \chi_i (K_{max} - K_{min}), && (4)\\ Y_i &:= Y_{max} - \chi_i (Y_{max} - Y_{min}), && (5) \end{aligned}$$

corresponds to the mortality, light capture capacity, and relative contribution to infiltration. The reader is referred to [FPB+25] for a detailed description of the model.

We also consider the single-species model

$$\begin{aligned} \partial_t B &= \Lambda W B - M B + l^2 D_B \partial_x^2 B, && (6a)\\ \partial_t W &= I H - L W - \Gamma W B + l^2 D_W \partial_x^2 W, && (6b)\\ \partial_t H &= P - I H + l^2 D_H \partial_x^2 H, && (6c) \end{aligned}$$

obtained from (1) by setting $N = 1$ (with $B := B_1$) and choosing a particular $\chi$ value, e.g. $\chi = 1$ for which $M = M_{min}$, $K = K_{min}$, $Y = Y_{min}$ and removing the mutation term (last term in (1a)). However, 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 (6) is a semi-parabolic reaction diffusion systems of PDEs, and similar models are treated as model problems in [Uec21], see also [Uec25] for demos.

**Remark 1.** a) Instead of discretizing in $\chi$ from the outset, one could interpret $\chi$ as a second continuous coordinate over $[0,1]$, leading to a nonlocal heterogeneous 2D problem for $B(\chi,x,t)$ with $\chi$–dependent coefficients in (1). However, since $W(x,t)$ and $H(x,t)$ remain independent of $\chi$, adopting the current formulation is conceptually and practically simpler, and this also reflects the ecological setting: Traits in vegetation models represent a discrete set of species characteristics, not continuous variations, and $D_\chi$ in (1a) is a mutation rate.
b) The figures in [FPB+25] are dense, often showing multiple solution branches, requiring long runtimes of the scripts, and generated partly with `pyplot`. Here we provide modified versions of the figures by simplified scripts to quickly compute subsets of the results in [FPB+25], and at the same time present some useful additional results which we believe illustrate some further features of the model.
c) All scripts can simply be run by calling them from the `Matlab` command line; however, we strongly recommend running them in cell-mode from the `Matlab` editor, to see the results of individual cells and commands. For easy online use, the folders `bwhsingle` for (1) and `bwhcom` for (6) both contain files `aaa.m` with file overviews similar to Tables 1 and 2. ⌋

# 2 Single species model.

The folder `single/` contains the files needed to compute a reduced version of the bifurcation diagram in [FPB+25, Fig.1] and of the Busse balloons in [FPB+25, Fig.4], see Table 1 for a summary.

Table 1: Scripts and functions in `bwhsingle/`

| file | purpose, remarks |
|---|---|
| `cmds1` | continuation of `hom` and branch switching to and continuation of Turing branches; also computation of localized snake, and some DNS to go to different wave numbers. |
| `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 `tint`. |
| `sG,sGjac` | rhs of (6), and Jacobian; these here have a simple standard structure. |
| `tint` | time integration (DNS) for (6) |
| `sgbra` | mod of library function `stanbra`; |

## 2.1 Basic branch continuation.

The function `bwhinit.m` initializes the problem, with the inputs 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 (`dir`). We put rather detailed comments into `bwhinit.m` and hence refer to the source for these. The basic procedure is: use `p = stanparam` to generate a `pde2path` problem structure and populate it with standard values for numerical parameters and controls (tolerances, switches for bifurcation detection method, etc, see `stanparam.m` for an overview). Several of these default values are subsequently overwritten to configure the specific problem setup. Here, e.g., by setting `p.fuha.outfu=@sgbra`, the branch output is handled by the custom function `sgbra`, a slight modification of the standard output function `stanbra`. Additionally, we create a standard 1D PDE object, which includes the finite element mesh and associated routines, and the initial solution vector is constructed by concatenating the field values and parameters: `p.u=[b0; w0; h0; par]`.

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.

```
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 1: `bwhsingle/oosetfemops.m`, precompute the FEM matrices.

Once the system parameters and main `pde2path` structure are set up, we can use them in `sG.m` to implement the rhs of our model (6).

```
function r=sG(p,u) % for bwh-single, hence standard RD system
n=p.np; b=u(1:n); w=u(n+1:2*n); h=u(2*n+1:3*n); % extract fields
par=u(p.nu+1:end); % extract parameters and give names to make code readable
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);

K=p.mat.K; M=p.mat.M(1:n,1:n); % stiffness and mass matrix
ov=ones(n,1);  % vector to simplify notation

Yi=Ymax+chi*(Ymin-Ymax); % dependent quantities
Mi=Mmax+chi*(Mmin-Mmax); Ki=Kmax+chi*(Kmin-Kmax); Lam=Lam0*ov*Ki./(b+Ki);
I=A*(Yi*b+f*Q)./(Yi*b+Q);  L=L0*ov./(1+R*b);
r1=-M*(Lam.*w.*b-Mi*b)+l^2*Db*K*b; % first compo of rhs
r2=-M*(I.*h-L.*w-Ga*w.*b)+l^2*Dw*K*w;
```

```
r3=-M*(pp-I.*h)+l^2*Dh*K*h;
r=[r1;r2;r3];              % full rhs
```

Listing 2: `bwhsingle/sG.m`, implementing the (negative ) rhs of model (6).

That's all we need if we use numerical Jacobians by setting up `p.sw.jac=0`; but for speed, it is recommended to implement the Jacobian

```
function Gu=sGjac(p,u) % Jac of single species sG
n=p.np; par=u(p.nu+1:end); Db=par(15); Dw=par(16); Dh=par(17);l=par(19);
[f1b,f1w,f1h,f2b,f2w,f2h,f3b,f3w,f3h]=njac(p,u); % derivative of nonlinearity
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)]];
Gu=kron([l^2*Db,0,0; 0,l^2*Dw,0; 0,0,l^2*Dh],p.mat.K)... % Gu=diffusion terms
    - p.mat.M*Fu;   %                                        -local terms
end
```

Listing 3: `bwhsingle/sGjac.m` (selection, see source for function `njac`), implementing $\partial_U G$, i.e., the Jacobian of $G$, with helper function `njac` (see source).

Finally, Listing 4 shows a modification of the standard simple linearly implicit time–stepper used for DNS for (6).

```
function [p,t1]=tint(p,t1,dt,nt,pmod) % mod of library function for DNS for
% semilinear RD systems. Linearly implicit, with prefactored Lam
% In : problem struct p as usual, t1=initial time, dt=stepsize, nt=#steps,
%      pmod=plotting interval (plot every pmod-th step)
% Out: (updated) p and t1
n=0; t=t1; Db=p.u(p.nu+15); Dw=p.u(p.nu+16); Dh=p.u(p.nu+17);
K=kron(diag([Db,Dw,Dh]),p.mat.K); % diffusion matrix
Lam=p.mat.M+dt*K; [L,U,P,Q,R]=lu(Lam); % prefactor stepping matrix
while(n<nt) % integration loop
  f=nodalf(p,p.u);   % the nonlinearity, i.e., everything except diffusion
  g=p.mat.M*p.u(1:p.nu)+dt*f;
  p.u(1:p.nu)=Q*(U\(L\(P*(R\g)))); n=n+1; t=t+dt; % time stepping:
  if(mod(n,pmod)==0); plotsol(p,p.plot.ifig,p.plot.pcmp,p.plot.pstyle); end
end
t1=t;
```

Listing 4: `bwhsingle/tint.m`, for DNS for (6); see source for `nodalf.m`.

In Listings 5–8 we provide (selections) from the main script `cmds1.m` that yields Fig.1 as a variant of [FPB$^+$25, Fig.1]. After initialization, we first continue the homogeneous steady state branch for stress-tolerant species with $\chi = 1$, which yields a number of (Turing) bifurcation points (BPs). We then do a branch switching to these Turing branches. Additionally, we compute a localized patterns "snake" [Kno15] bifurcating from the first Turing branch, we generate additional solutions by direct numerical simulation (DNS). Then plotting.

```
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];
b0=5.25; w0=3.15; h0=pp*(Yi*b0+Q)/(A*(Yi*b0+f*Q));  % initial guess
dir0='s1'; dir=[dir0 '/hom']; % % Output folder, chi=1
%% === 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;   % Eigenvalue reference for eigs routine
```

```
p.nc.neig=10;         % Number of eigenvalues to compute
p.sol.ds=-0.1;      % Initial cont.step size (negative=backward in parameter)
p=cont(p,60);       % Continue for 60 steps along homogeneous branch
```

Listing 5: bwhsingle/cmds1.m (part): initialization, and continuation of hom branch.

```
%% === Branch switching to Turing T1
p=swibra('s1/hom','bpt1','s1/T1',0.001); % branch-switching at 'bpt1' in s1/hom
p.nc.neig=6;    % #eigenvalues to compute; here rather few for speed
p.nc.eigref=-3;% ref.eigenval for eigs; here rather negative to not miss instab
p.nc.dsmin=1e-6;% min cont. step size; quite small since difficult folds
p.nc.dsmax=5; p0.nc.dlammax=4;  % max arclength and parameter step sizes
p.nc.tol=1e-9;      % tolerance for Newton's method during continuation
p=cont(p,200); % go!
%% === Branch switching to Turing T2, T3 and T4
p=swibra(dir,'bpt2','s1/T2',0.1); p.nc.neig=5;p.nc.eigref=-3;p.nc.dsmin=1e-6;
p.nc.dsmax=5; p.nc.dlammax=4; p=cont(p,120);
```

Listing 6: bwhsingle/cmds1.m (part): branch switching to first Turing branch T1, and similarly to T2.

```
%% DNS to k=0.28 branch,preparation
p=loadp('s1/T1','pt170','du'); p.u(p.nu+1)=180;  % change P
t1=0; dt=0.01; nt=50000; pmod=100; % #int.steps and plot each pmod-th step
%% repeat until r=1e-6
[p,t1]=tint(p,t1,dt,nt,pmod); res=pderesi(p,p.u); r=norm(res,'inf')
%% continue: reset counter and set filename to T2-double
p=resetc(p); p=setfn(p,'s1/T2d'); p=cont(p,100);
%% continue in other direction
p=loadp('s1/T2d','pt10','s1/T2db'); p.sol.ds=-p.sol.ds; p=cont(p,100);
%% DNS to k=0.07 soln (single hump); but cannot continued due to transl.invar
p=loadp('s1/T2db','pt20','single'); p.u(p.nu+1)=155;
p.u(1:round(p.np/4))=0; plotsol(p); title('one patch deleted'); pause
t1=0; dt=0.01; nt=10000; pmod=100;
[p,t1]=tint(p,t1,dt,nt,pmod); res=pderesi(p,p.u); r=norm(res,'inf')
fig(6); title(['t=' mat2str(t1,3)]);
```

Listing 7: bwhsingle/cmds1.m (part): DNS from T1/pt170 (unstable) to $k = 0.28$ (4 patches) branch, and continuation of that into positive and negative $P$ direction. Then DNS to a single patch branch by deleting a patch from a 4-patch solution.

```
%% === Plot branches; see plotbra documentation for arguments of plotbra
cHom=[0.39,0.83,0.07]; cT1=[0.40,0.53,0.25]; % colors
cT2=[0.64,0.81,0.43]; cT1s=[0.13,0.7,0.67]; cT028=[0.13,0.9,0];
c=1; ylab='||B||'; % select compo (see sgbra) and label to plot
%c=3; ylab='max(B)'; % (uncomment as desired)
mclf(3); plotbra('s1/hom','cl',cHom,'tyun','--','cmp',c,'lab',10,'ln','hom');
plotbra('s1/T1','cl',cT1,'tyun','--','cmp',c,'lab',[100,170],'ln',{'A','B'});
```

```
%% === soln plots
plotsol('s1/T1','pt100'); title('A'); pause;
plotsol('s1/T1','pt170'); title('B'); pause;
```

Listing 8: bwhsingle/cmds1.m, end: branch and solution plotting.

As already said (Remark 1c), we strongly recommend loading cmds1.m into the Matlab editor, look at the further comments we give there, and then run this file "cell–by–cell", i.e., execute individual blocks and advance. However, you can also simply type cmds1 at the command line; this yields the following (initial) output (with altogether eight BPs between $P = 300$ and $P = 250$):

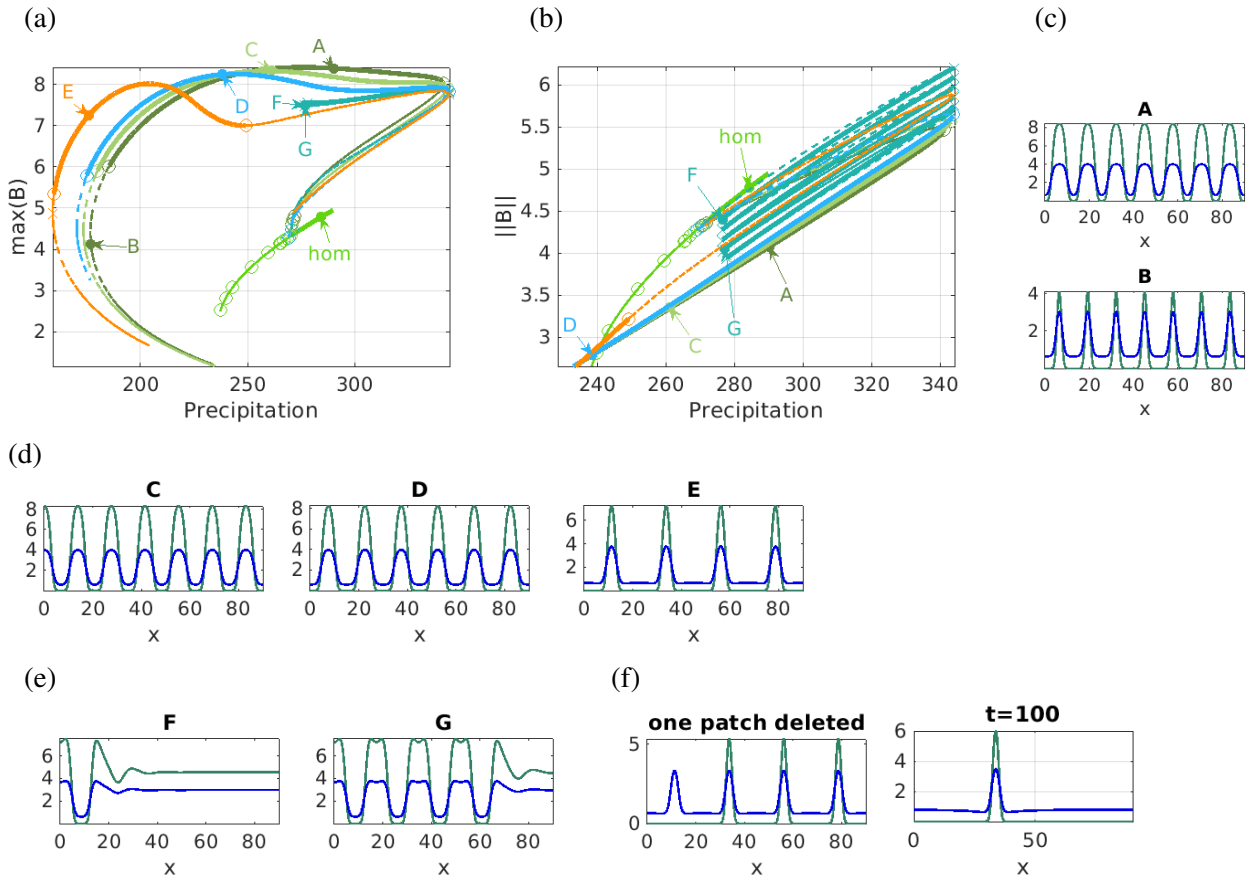


Figure 1: Results from cmds1.m for the single-species model (6) with $\chi = 1$. (a) Bifurcation diagram of Turing branches, $\max(B)$ over $P$, and secondary branch of localized patterns. (b) Same data as in (a) but $\|B\|$ over $P$, where the snake of localized patterns becomes visible. (c) Samples from the primary Turing branch, wave number $k = 0.489$ (7.5 waves), A stable, B unstable. (d) further patterns with $k = 0.454$ (C), $k = 0.419$ (D), and $k = 0.28$ (E). This last one was obtained by running DNS from B, and then running continuation again. (e) Samples from the snake. (f) Deleting one vegetation patch from the $k = 0.28$ patch at $P = 155$ and running DNS; this yields convergence to a single patch, which subsequently slowly moves to the middle. The associated branch can, however, not be easily continued due to the "almost" translational invariance, as the Neumann BCs are no longer numerically sufficient to fix the position.

```
    >> 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 0.000244141
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
  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
```

## 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 of the solution set. 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 useful for mapping stability boundaries and organizing bifurcation diagrams. These techniques are standard in numerical bifurcation software (e.g., pde2path in our case), see, e.g., [Uec21]. The numerical continuation of BPs is based on the extended system

$$H(U) = \begin{pmatrix} G(u,\lambda) + \mu M_d \psi \\ G_u^T(u,\varphi)\Psi \\ \| \psi \|_2^2 - 1 \\ \langle \psi, G_\lambda(u,\varphi) \rangle \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad U = (u,\psi,\varphi), \tag{7}$$

Here, $(u,\lambda)$ denotes a simple branch point (BP) for continuation in the primary parameter $\lambda$, and $\psi$ 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.

One main issue is how to compute $\partial_u(G_u^T\psi)$, which is the main new part in the Jacobian of the extended system $H$. For simple RD systems, this can often be easily implemented, but we can also make our life simple and set p.sw.spjac=0 to let `pde2path` approximate $\partial_u(G_u^T\psi)$ numerically; although, for accuracy and precision it is recommended to provide it explicitly (given by bpjac.m and activated by p.sw.spjac=1). With this, Listing 9 shows the main part of cmds2.m, which computes and plots the upper part of the BB for (6) via BPC. In detail, the primary Turing branch `s1/T6` with wave number $k_0 = 0.489$ shows a sideband instability at BP5. Now continuing this BP in $P$ and the scaling parameter $\ell$ yields the values of $P$ at which the Turing pattern with wave number $k = k_0/\ell$ becomes unstable, and hence a part of the boundary of the Busse balloon (BB) of stable wave number — $P$ combinations, see Fig.2. To compute further parts of the BB, the procedure must be repeated for other starting wave numbers $k_0$, which, after some lengthy scripting, yields [FPB+25, Fig.4].

```
%% === Init: continue 'bpt5' on branch s1/T1 wrt to parameters 1 and 19
%  - 1=precipitation (P)
%  - 19=l (domain length rescaling or trait scaling)
% Output folder: 's1/bpa'
p=bpcontini('s1/T1','bpt6',19,'s1/bpa',1e-3); % Start from BP5
p.plot.bpcmp=5;      % Plot 5th component during continuation
p.nc.dsmax=0.1; p.nc.dsmin=1e-5;  % Max and min step size
p.nc.del  =0.2;     % finite difference discretization of extra terms
p.nc.lammin=0;       % Lower bound for continuation parameter
p.sw.spcalc=0; p.sw.bifcheck=0; % Disable spectral comp and bif detection
p.nc.tol=1e-8;        % Newton tolerance
p.sw.spjac=1;         % use analytic for 2nd directional derivatives
hucl; p=cont(p, 60);  % clear windows and run
%% === Reverse Continuation ===
p=loadp('s1/bpa','pt0','s1/bpb'); % Load initial point from bpa
p.sol.ds=-1.3*p.sol.ds; p=cont(p,20); % Reverse direction and go
```

Listing 9: bwhsingle/cmds2.m (main part, rest is plotting).

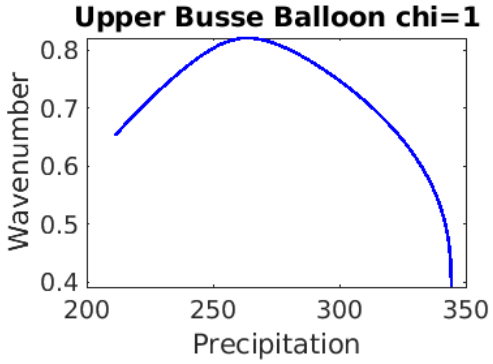


Figure 2: single/cmds2.m; Busse balloon boundary computation from bifurcation point continuation.

# 3 Community model: pde2path implementation.

The pde2path implementation of the community model (1) deviates from that of standard reaction–diffusion systems due to the $N$ biomass fields $B_1,\dots,B_N$ with (in principle) arbitrary (non–small) $N$, and requires some adjustments:

- Additional effort is needed to implement the right-hand side $-G(u)$ of (1) in the function sG.m (Listing 10), 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.
- We need a user–provided plotting function userplot.m, where we also wish to display spatial averages

$$\langle B(\cdot,\chi)\rangle := \frac{1}{|\Omega|}\int_\Omega B(\chi,x)\,\mathrm{d}x, \tag{8}$$

  written formally for $B_i(x) = B(\chi_i,x)$, i.e $\langle B_i\rangle = \frac{1}{|\Omega|}\int_\Omega B_i(x)\,\mathrm{d}x = \frac{1}{1x}\sum_{j=1}^{n_p} B_i(x_j)$ after discretization in $\chi$ and $x$. Therefore we set p.plot.pstyle = -1 such that the internal plotting function plotsol directly calls userplot. Here, to facilitate switching between different plotting styles, userplot is just an interface to plotting functions (uplot1, uplot2, or uplot3), controlled by p2pglob.ps, where p2pglob is a global variable.[1]
- Some minor modifications are also applied to the library function tintxs, see Listing 11, which is used for direct numerical simulations (DNS), both for generating initial guesses for continuation and for stepping through the Busse balloon.

However, the general data layout remains the same: we define the solution vector $u = [B_1,\dots,B_N, W, H, \mathtt{pars}]^T$, where all biomass and water components are discretized on np grid points over a 1D domain of length 1x, and par holds the model parameters. Alltogether, this results in p.nu=(N+2)*p.np degrees of freedom (DoF), arising from the concatenation of the $N$ biomass fields, and $W$ and $H$ fields. Table 2 provides an overview of the used files.

[1] Since global variables are potentially dangerous, p2pglob with subfields is the recommended name for the *only* global variable in pde2path.

Table 2: Scripts and functions in bwhcomm.

| file | purpose, remarks |
|---|---|
| cmds1 | script for computation of steady state bifurcation diagrams for (1), also again including some DNS to get near particular steady states. |
| cmds2 | script for Busse balloon computations based on results from cmds1. |
| 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 | branch output function, here also providing $\chi_{max}$. |
| userplot | modified plotting function; called from plotsol if p.plot.pstyle=-1 (which we set in bwhinit), and further controlled by p2pglob.ps to conveniently switch between uplot1 (only $B = B_1, \ldots, B_N$), uplot2.m ($B$ and $W$), and uplot3 ($B, H$, and $W$). |
| tintxs | semi-implicit time stepper. |
| bfbb | Brute-force Busse balloon function. Extract stability from precomp. branches. |
| bbdns | function for DNS with varying precipitation. |
| plotds | function to visualize the trajectories within the Busse balloon. |

For clarity, the rhs $-G(u)$ of the model (1) reads, component wise,

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

where $i = (1, \ldots, N)$, and Listing 10 shows the implementation. This follows the same general principles as bwhsingle/sG:

- extract the fields $B_i, W, H$ and parameters from u;
- implement the "nonlinearity" (everything except spatial diffusion);
- finally compose r=sG(p,u).

We comment on some details: For convenience (easy loops) we reshape (l2) the vector $B = (B_1, \ldots, B_N) =$ u(1 : N∗n) into a $n \times N$ matrix; the "nonlinearity" includes the diffusion in $\chi$ (with BCs in $\chi$), see line 21–25. From the implementation of sG, the Jacobian follows in a lengthy but rather straightforward way, and we refer to the source bwhcom/sGjac.m. The initialization file bwhinit.m is quite similar to the single species model (6), except that we need to initialize the N fields in a $\chi$-dependent way, and as explained above, we set p.plot.pstyle=-1 for plotting.

```
1  function r=sG(p,u) % for bwhcom. N b-fields, hence loop over i=1..N;
2  N=p.N; n=p.np; b=reshape(u(1:N*n),n,N);     % b
3  w=u(N*n+1:(N+1)*n); h=u((N+1)*n+1:(N+2)*n); % w and h; next extract parameters
4  par=u(p.nu+1:end); pp=par(1);Lam0=par(2);Ga=par(3);A=par(4);R=par(5);L0=par(6);
5  f=par(7); Q=par(8); Kmin=par(9); Kmax=par(10); Mmin=par(11); Mmax=par(12);
6  Ymin=par(13); Ymax=par(14); Db=par(15); Dw=par(16); Dh=par(17); Dchi=par(18);
7  chimin=par(19); chimax=par(20);
8  K=p.mat.K; M=p.mat.M(1:n,1:n); % (spatial) Laplacian and Mass-matrices
9  bt=zeros(n,1);btt=bt;r=zeros(N*n,1); % allocate b-tilde, b-tilde-tilde, and r
10 chii=linspace(chimin,chimax,N); dchi=chii(2)-chii(1); ov=ones(n,1);
11 for i=1:N % fill bt and btt
12     bt=bt+b(:,i); Yi=Ymax+chii(i)*(Ymin-Ymax); btt=btt+b(:,i)*Yi;
13 end
```

```
I=A*(btt+f*Q)./(btt+Q); L=ov*L0./(1+R*bt);  % infiltration and loss
for i=1:N
    Ki=Kmax+chii(i)*(Kmin-Kmax); Mi=Mmax+chii(i)*(Mmin-Mmax);
    Lami=Lam0*Ki./(bt+Ki);
    bi=b(:,i);
    switch i  % chi-diffusion terms, i=1 and i=N with Neumann BCs
        case 1; bcc=(b(:,2)-2*b(:,1))/dchi^2;
        case N;  bcc=(-2*b(:,N)+b(:,N-1))/dchi^2;
        otherwise; bcc=(b(:,i+1)-2*b(:,i)+b(:,i-1))/dchi^2;
    end
    r1=-M*(Lami.*w.*bi-Mi*bi+Dchi*bcc)+Db*K*bi;
    r((i-1)*n+1:i*n)=r1;
end
r2=-M*(I.*h-L.*w-Ga.*bt.*w)+Dw*K*w;  % W-eq
r3=-M*(pp-I.*h)+Dh*K*h;              % H-eq
r=[r;r2;r3];                         % assemble full rhs
```

Listing 10: bwhcom/sG.m, rhs for (1). Some nonstandard procedures are the reshaping of u(1:N*n) into a matrix in l2, and the discrete $\chi$–diffusion in l19-l23.

```
function [p,t1,ts,nc]=tintxs(p,t0,ts,dt,nt,nc,pmod,smod,nffu,varargin)
% TINTXS: time integration with time-series output, LU-decomp. of M+dt*K,
%  Here modified to only treat x-diffusion implicitly, chi-diff expl.
par=p.u(p.nu+1:end); Db=par(15); Dw=par(16); Dh=par(17); N=p.N; n=p.np; ng=N*n;
r=norm(resi(p,p.u),'inf'); ts=[ts [t0;r]]; % put time and residual into ts
K=p.mat.K; J=sparse(ng); % compute "Jacobian" for the implicit part
for i=1:N; J((i-1)*n+1:i*n,(i-1)*n+1:i*n)=Db*K; end  % diffusion B
J(N*n+1:(N+1)*n,N*n+1:(N+1)*n)=Dw*K; % diffusion w
J((N+1)*n+1:(N+2)*n,(N+1)*n+1:(N+2)*n)=Dh*K; % diffusion h
Lam=p.mat.M+dt*J; % mclf(20); spy(Lam); pause
[L,U,P,Q,R]=lu(Lam); t=t0; n=0;
while(n<nt) % integration loop
  f=nffu(p,p.u); % "nonlinearity"=everything but diffusion, here with chi-diff
  g=p.mat.M*p.u(1:p.nu)+dt*f;
  p.u(1:p.nu)=Q*(U\(L\(P*(R\g)))); t=t+dt;  n=n+1;
  if(mod(n,pmod)==0);  % plotting and time-series output.
      r=norm(resi(p,p.u),'inf'); ts=[ts [t;r]]; % put time and residual into ts
      tits=['t=' mat2str(t,4) ', r=' mat2str(r,3)];
      plotsol(p,p.plot.ifig,p.plot.pcmp,p.plot.pstyle);
      title(['u_1, ' tits],'fontsize',12); set(gca,'fontsize',12); drawnow;
  end
  if(smod~=0 && mod(n,smod)==0);  stansavefu(p); end % save soln
end
t1=t; nc=nc+nt;
```

Listing 11: bwhcom/tintxs.m, DNS for (1), treating spatial diffusion implicitly, everything else explicitly.

## 3.1 Basic bifurcation diagram

The script cmds1.m, see Listing 14 and Fig.3, which essentially corresponds to [FPB+25, Fig.3], computes a number of Turing bifurcations from the homogeneous steady state branch, a secondary bifurcation to a snake of localized states, and also computes some further Turing states via deleting patches from a given Turing pattern, running DNS until convergence to a new Turing pattern, which is continued by steady state continuation. This yields, e.g., state C from A, and state D from B. Similarly, to get to the initial spatially homogeneous state, we use a rough initial guess followed by DNS.

Additionally, to compute the branches with states E and F, we apply a "standard" trick when dealing with many BPs (here on the hom branch) close together: using, e.g., p0=cswibra('1/hom','bpt2',aux) we compute aux.m many "kernel vectors", i.e., the eigenvectors to the aux.m many smallest eigenvalues. Subsequently, we can choose any (linear combination) of these kernel vectors, and try to continue "in that direction". For BPs close together (or BPs of higher multiplicity), this generally works and is much more efficient than localizing all of the BPs individually by using a very small continuation stepsize ds, and then doing individual `swibra`'s.[2]

Altogether, the script cmds1.m for computing the shown Turing branches and the snake of localized patterns (with 1000 continuation steps) needs about 2h runtime.

```
p=bwhinit(lx,nx,N,par,dir,aux); p=setfn(p,dir);plotsol(p);
%% DNS to get near spatially homogeneous steady state
t1=0; ts=[]; dt=2e-3; nt=6e4; nc=0; pmod=nt/210; smod=pmod;
[p,t1,ts,nc]=tintxs(p,t1,ts,dt,nt,nc,pmod,smod,@sGdns);
%% Newton loop for steady state
p.nc.almin=0.25;p.nc.tol=1e-10; [p.u,p.r,iter]=nloop(p,p.u);
fprintf('res=%g, iter=%i\n',norm(p.r,Inf),iter); plotsol(p);
%% continue hom branch
p.sw.para=2; p.nc.eigref=-1e-1; p.nc.tol=3e-10; p.nc.neig=5;
tic; p=cont(p,15); toc % large steps
p.sol.ds=-0.1; p.nc.dsmax=2; p=cont(p,20); %smaller steps for BP detection
%% swibra to first Turing branch T1
p=swibra('1/hom','bpt1','1/T1',0.001); p.nc.dsmin=1e-6; p=cont(p,10);
p.sw.bifcheck=0; p.nc.eigref=-4; p.nc.dsmax=10; p.nc.neig=2; p=cont(p,100);
%% swibra to primary snake of localized patterns
p=swibra('1/T1','bpt1','1/T1-1',0.001); pause; p.nc.tol=1e-8; p=cont(p,10);
p.file.smod=50; p.sw.bifcheck=0; p.nc.dsmax=5; p.nc.eigref=-4; p=cont(p,1000);
```

Listing 12: bwhcom/cmds1.m (start); initialization, DNS to hom, continuation of hom, swibra to and cont of T1, and swibra to and cont of localized snake.

```
p.sw.bifcheck=0; p.nc.eigref=-4; p.nc.dsmax=10; p.nc.neig=2; p=cont(p,100);
%% cswibra to T2C, k=0.66
aux=[]; aux.m=4; aux.besw=0;p0=cswibra('1/hom','bpt2',aux); p0.sw.spcalc=1;
p0.sw.bifcheck=2; p0.nc.eigref=-3; p0.file.smod=20; p0.nc.neig=4;
p0.nc.dsmin=1e-4; p0.nc.tol=1e-9; p0.nc.bisecmax=10; p0.nc.dsmax=1;
%% gentau
p=gentau(p0,[0 0 1]); p=setfn(p,'1/T2C'); p.sol.ds=0.1; plotsol(p);
p.sw.bifcheck=0; p.nc.eigref=-4; p.nc.dsmax=10; p.nc.neig=2; p=cont(p,100);
```

Listing 13: bwhcom/cmds1.m, cswibra at hom/bpt2 and continuation of T2C.

```
p.sw.bifcheck=0; p.nc.eigref=-4;p.nc.dsmax=10;p.nc.neig=2;p=cont(p,100);
%% "delete patches" to go to other k via DNS
p=loadp('1/T1','pt70','du'); b=p.u(1:p.np*p.N); M=40; % how much to delete
for i=1:p.N; b((i-1)*p.np+p.np-M:i*p.np) = 0; end
p.u(1:p.N*p.np)=b; w=p.u(p.N*p.np+1: (p.N+1)*p.np); wmin=min(w);
w(p.np-M:end)=wmin; p.u(p.N*p.np+1: (p.N+1)*p.np)=w;
h=p.u((p.N+1)*p.np+1: p.np*(p.N+2)); hmax=max(h);
h(p.np-M:end)=hmax; p.u((p.N+1)*p.np+1: p.np*(p.N+2))=h; plotsol(p);
% DNS
t1=0;  ts=[]; dt=0.5e-2; nt=1e5; nc=0; pmod=nt/200; smod=pmod;
[p,t1,ts,nc]=tintxs(p,t1,ts,dt,nt,nc,pmod,smod,@sGdns);
```

[2]However, the order in which multiple kernel vectors are computed (by inverse vector iteration) may be somewhat machine dependent. Therefore, we recommend inspecting the plots of the kernel vectors and, if necessary, modifying the generation of the bifurcation direction via `p=gentau(p0, coeff)` by choosing an appropriate coefficient vector `coeff`; see sources for more comments.

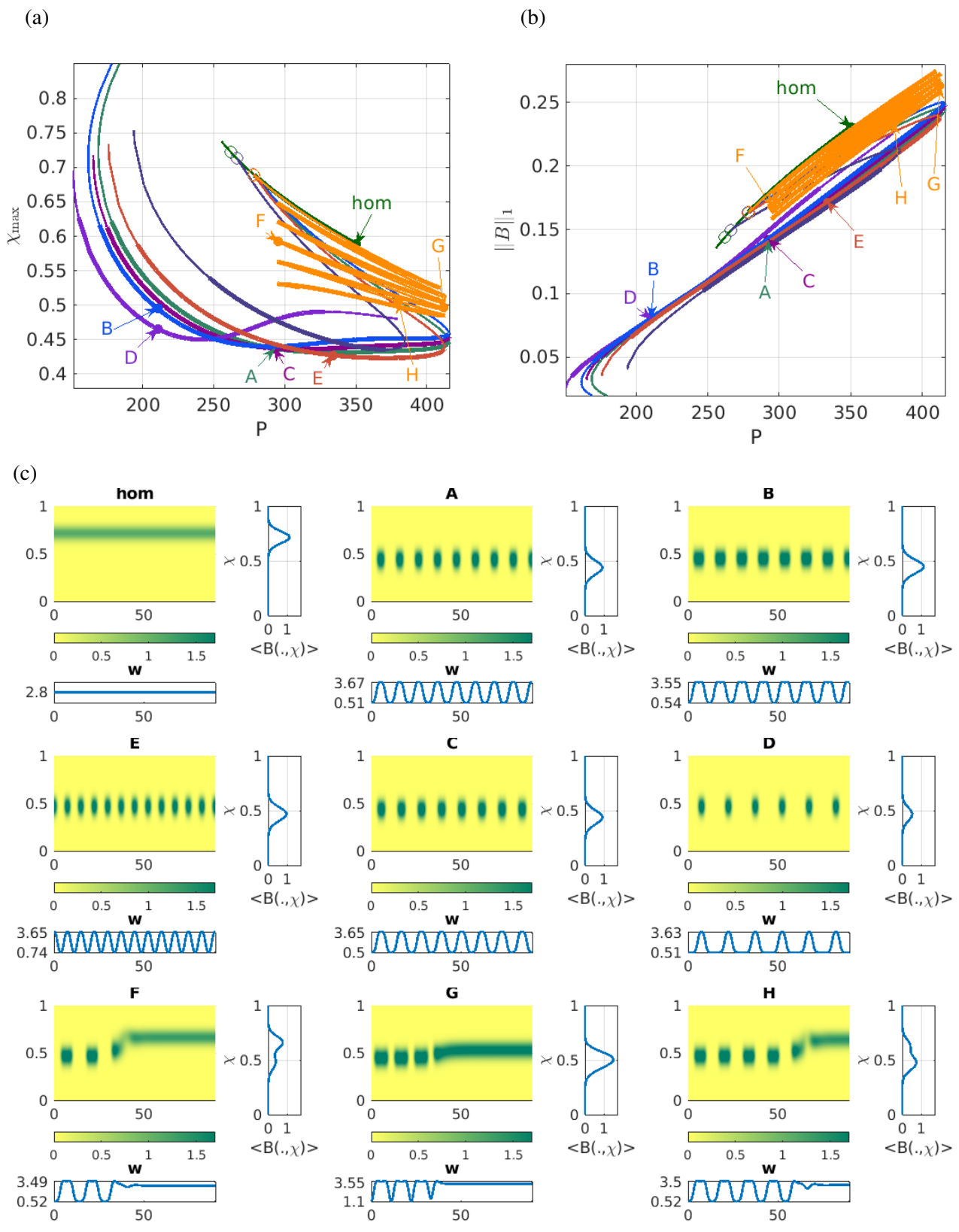


Figure 3: Results from bwhcom/cmds1.m for (1). (a,b) BDs with $\langle \chi_{\max} \rangle$ and $\|B\|_1$ over (P). (c) Samples: the wave numbers on the Turing branches are: $k = 0.59$ (A), $k = 0.52$ (B), $k = 0.55$ (C), $k = 0.42$ (D), $k = 0.84$ (E). In the snake (orange, with samples F, G, H), $\langle B \rangle$ (see (8)) alternates between unimodal and bimodal distributions.

```
%% Newton loop for steady state
p.nc.almin=0.25;p.nc.tol=1e-10; [p.u,p.r,iter]=nloop(p,p.u);
fprintf('res=%g, iter=%i\n',norm(p.r,Inf),iter); plotsol(p);
%% cont steady state
p=setfn(p,'1/T1b'); p=resetc(p); stansavefu(p); p.sol.ds=-0.01; p=cont(p,41);
%% reverse direction
```

Listing 14: bwhcom/cmds1.m, artificially "dropping" patches and using DNS to go to longer wavelength pattern. The remainder of the script uses similar commands to compute further branches, and deals with plotting.

## 3.2 Busse balloon computations

Busse balloon diagrams can be computed via bifurcation point continuation, as in §2.2, but this did not converge robustly for the community model, and thus we use a brute-force approach and extract the stable regions from the already computed branches, see Listing 15, and Fig.4(a).

**Remark 2.** The BB sampling from the relatively few branches computed in cmds1.m is rather coarse. Figure 4(b) shows the BB sampling from the data used in [FPB+25] for Fig.4 and Fig.5; there we computed many more branches, on a domain of length 140, hence yielding a finer wave number sampling, but also requiring more DoF for the spatial discretization. Altogether, this results in a long script with total runtime of more than 10h for computing the Turing branches. ⌋

```
function [kv, pv]=bfbb(kv,pv,dir,k0) % Brute Force Busse Ballon
% scans branch in dir for stable solns and returns the par-values
% wave-nr k user supplied!  (could be obtained from soln via FFT,
% but here we keep life simple)
p=loadpp(dir); pv0=p.branch(4,:); inv=p.branch(3,:); % take data from branch
lb=length(pv0);
for i=1:lb;    % if stable, then add point (and wave-nr) to list
    if inv(i)<1; kv=[kv k0]; pv=[pv pv0(i)]; end
end
```

Listing 15: bwhcom/bfbb.m Brute force Busse balloon function, scanning precomputed branches for stable solutions.

```
kvcm0=[];  % Wave numbers
pvcm0=[];  % Precipitation P values
% List of folders and associated wavenumbers
branchList={'1/T1',0.59; '1/T2',0.52; '1/T1b',0.55; '1/T1bb',0.55;
    '1/T2b',0.42; '1/T2bb',0.42; '1/T2C',0.664; '1/T3D',0.838};
for i=1:size(branchList, 1) % Loop through list and collect stable solutions
    dir=branchList{i, 1}; k0 =branchList{i, 2};
    [kvcm0, pvcm0]=bfbb(kvcm0, pvcm0, dir, k0);
end
%% === Plot Busse Balloon
mclf(10); hold on;
plot(pvcm0, kvcm0, 'x', 'Color', 'm');  % Stable points from sampling
```

Listing 16: bwhcom/cmds2.m (main part). Busse balloon sampling from precomputed branches.

**DNS in and near the Busse balloon.** In [FPB+25, Fig.5] we discuss patch thinning and patch dilution under decreasing precipitation in and near the Busse balloon (BB). This is (like Fig.4(b)) again based on a large domain with lx=140 and hence on lengthy computations and long scripts. In bwhcom/cmds2 we continue explaining the method based on the simplified setting with lx=90. The

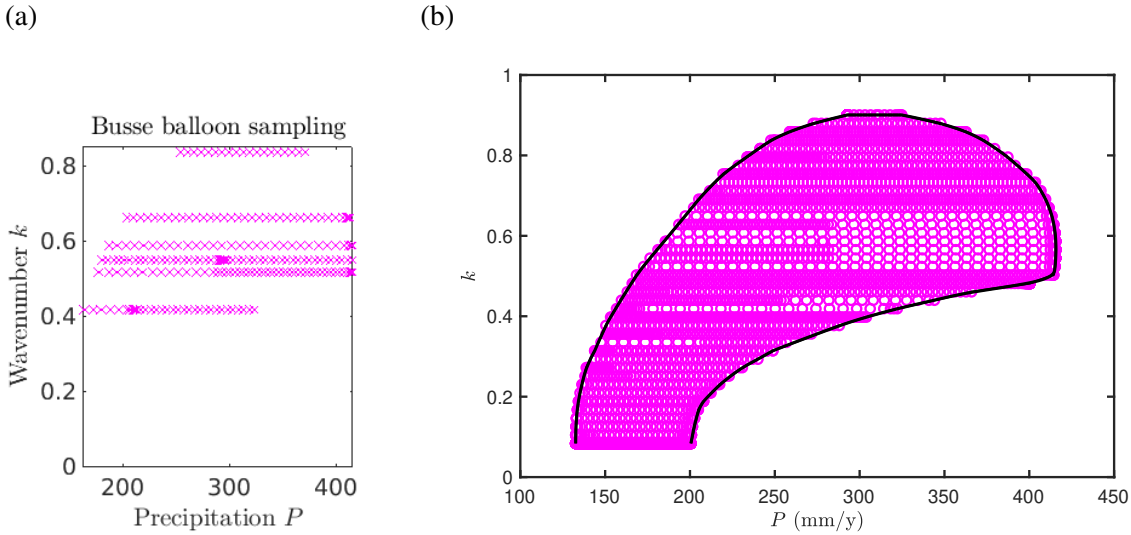


Figure 4: Brute force Busse balloon computations. (a) Results from cmds2.m based on the branches computed in cmds1.m. (b) Same for more branches on a longer domain, see [FPB+25], with post-processing to extract the boundary of the BB (thick black line).

basic idea is as follows: We start with some stable pattern inside the BB from a precomputed branch with wave number $k$ at some $P$, and then iterate the following process:

> Change $P$ (by some increment p.incr), and apply a perturbation to the state $u = (B,W,H)$; run DNS for time p.T; at the end, extract and store the wave number $k$ of the current state.
> Repeat for a prescribed number of steps, or until the solution collapses to bare soil, or goes to spatially uniform vegetation. (9)

After finishing (9) we plot the data obtained as in Fig.5, where $(P,k)$ show the path through (or near) the BB, and the colors can illustrate $\chi_{\max}$ or $\|B\|_1$.

Naturally, the results depend strongly on the rate of change of $P$, and weakly on the details of the extraction of the wave number $k$, and the way of perturbation of $u$. If we do not apply any perturbations, then the solutions should still slide along the boundary of the BB due to instability triggered by roundoff errors in the DNS, but that would require exceedingly long time scales. Here we perturb $H$ (the surface water), which can be thought of as random perturbations in rainfall. However, for definiteness, instead of applying random perturbations (at fixed instants in time), we perturb $H$ according to

$$H_{\text{perturb}}(x) = (1 + a \sum_{k \in k_{\text{pert}}} \cos(kx)) H(x), \tag{10}$$

where for the perturbation wave numbers we choose $k_{\text{pert}} = \{1/2, 1, 3, 5\}$, and for the perturbation amplitude $a = 0.01$. The obtained results do depend on these choices, but rather weakly, and, as discussed in [FPB+25], the rate of change of $P$ (p.incr/p.T in the discrete setting) is what matters.

Listing 17 shows how (9) is implemented, Listing 18 shows the pertinent part from cmds2, and Fig.5 shows some results, for p.T=1 and p.$incr = \pm 1$, where $-1$ means decreasing $P$. We then drop to bare soil at $P \approx 130$, and subsequently we reload a point before dropping to bare soil and increase $P$ again (p.$incr = 1$) to go through the lower part of the BB. Again, these scripts are meant to explain the method and general setup, and to yield basic results in moderate runtimes, and the results in [FPB+25, Fig.4 and Fig.5] require very long scripts and runtimes.

```
function [p,ds]=bbdns(p,ds,nst) % Busse balloon DNS with varying P
P=p.u(p.nu+1); N=p.N;n=p.np;nx=n; % extracting data from p: prec.P and dims
x=getpte(p); x=x'; lx=max(x); xn=2*pi*x/lx; % space
dt=0.0075; nt=floor(p.T/dt); pmod=nt/10; smod=0;
for i=1:nst
 P=P+p.incr; p.u(p.nu+1)=P; t1=0; ts=[]; % change P, prepare DNS
 pert=0*xn; pwn=[0.5 1:2:5]; npwn=length(pwn); pav=p.pa*[1:npwn];
 for l=1:npwn; pert=pert+pav(l)*cos(pwn(l)*xn); end % for pertubing H
 p.u((N+1)*n+1:(N+2)*n)=p.u((N+1)*n+1:(N+2)*n).*(1+p.pa*pert);
 nc=0; [p,t1,ts,nc]=tintxs(p,t1,ts,dt,nt,nc,pmod,smod,@sGdns); % DNS
 out=sgbra(p,p.u); % output at end of DNS; now extract further data, i.e.,
 % wave number k of patterns, based on H, in different ways
 h=p.u(n*(N+1)+1:n*(N+2)); hdiff=max(h)-min(h);
 if hdiff<1e-2; hdiff, break; end % stop if soln is bare soil or spatially hom
 threshold=min(h)+p.threshp*max(h); % extract k by counting patches (via H)
 above_threshold=abs(h)>threshold;
 start_patches=find(diff([false; above_threshold(:)]) == 1);
 km2=2*pi*(length(start_patches)/lx);
 h=h-mean(h); hh=abs(fft(h));hh=movmean(hh,3);hh=hh(1:floor(nx/2)+1); % use FFT
 [hmax,idx]=max(hh(1:end)); % find argmax(FFT(h))
 fs=nx/p.vol; fr=(0:floor(nx/2))*(fs/nx); fr1=fr(1:end); % FFT scaling
 try; km=2*pi*fr1(idx); catch km=1; end  % extract k
 bt=out(1); chi=out(3); P, ds=[ds [P;bt;chi;km;km2]]; % store data
 p.file.count=p.file.count+1;
 if mod(i,10)==0; p.fuha.savefu(p); end  % save state (for plotting or reload)
end
```

Listing 17: bwhcom/bbdns.m, for DNS through Busse balloon with varying $P$.

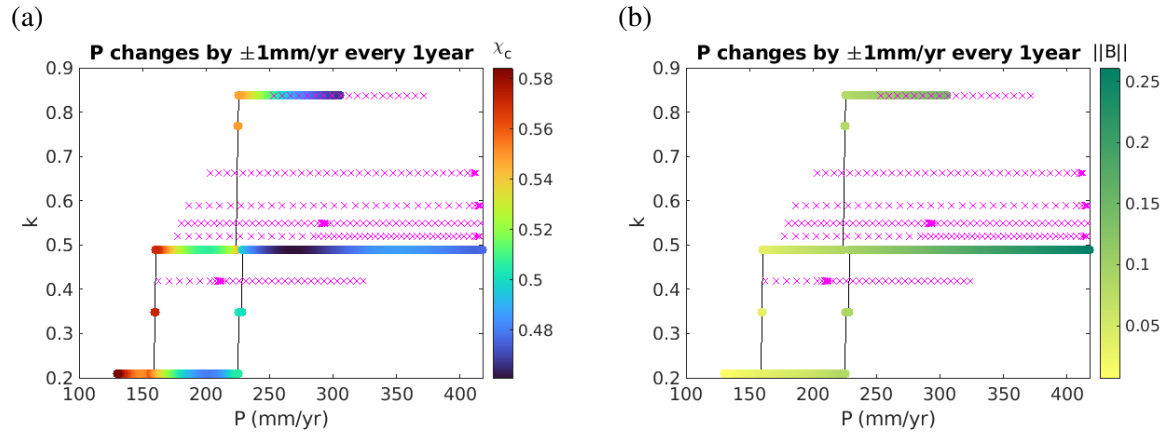


Figure 5: Further results from cmds2.m; stepping through Busse balloon (overlaid in magenta); color chosen from $\chi_{\text{center}}$ (a), and from $\|B\|_1$ (b).

```
%% === DNS through BB with change of P
p=loadp('1/T3D','pt60','1/1yra'); % load start point and set up output folder
p.u(p.nu+1)=230; % set P to some convenient starting value
p.incr=-0.5; p.T=1;  % increment and time-interval in precipitaion.
p.threshp=0.1;  % threshold to define deviation from baresoil, i.e., patches
nst=400; p.file.count=0;  % #time steps, and counter for saved output files
p.pa=0.05;          % amplitude of perturbations in H
ds1=[];             % for output data
[p,ds1]=bbdns(p,ds1,nst); % go! (repeat if useful)
%% backward: load point before dropping to bare soil, reverse rate
```

```
30 dir='1/1yra'; lastpt=['pt' mat2str(max(getlabs(dir))-10)];
```

Listing 18: bwhcom/cmds2.m (continued), DNS through BB, remainder is plotting.

# References


[FPB+25] M. Ferré, I. Pavithran, B. K. Bera, H. Uecker, and E. Meron. Vegetation pattern formation and community assembly under drying climate trends. *In review*, 2025.

[Kno15] Edgar Knobloch. Spatial localization in dissipative systems. *conmatphys*, 6(1):325–359, 2015.

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

[Uec25] H. Uecker. pde2path – a matlab package for continuation and bifurcation in systems of pdes, v3.1, 2025.