Numerical continuation and simulation for a spatial vegetation-water-herbivore system

Joydeep Singha[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: joydeepsingha105@gmail.com, August 11, 2025

Abstract

We describes the codes for the manuscript *"Traveling vegetation–herbivore waves may sustain ecosystems threatened by droughts and population growth"* by Singha et al. The methods include direct numerical simulation using `py-pde`, and numerical continuation using `pde2path`.

1 Brief description of the model

In [SUM25] we consider a spatially explicit model for the coupled dynamics of vegetation, soil water, and herbivores in dryland ecosystems. The model captures two critical stressors—limited water availability and herbivory—and incorporates feedback mechanisms that can give rise to spatial self-organization. In particular, the model accounts for a behavioral component of herbivore movement, referred to as "vegetaxis," in which herbivores are attracted to denser vegetation patches. This feature plays a crucial role in shaping vegetation–herbivore patterns.

Let $B(x,t)$, $W(x,t)$, and $H(x,t)$ represent the vegetation biomass density, soil water content, and herbivore biomass density, respectively, at spatial location x and time t. The governing equations are

$$\partial_t B = \Lambda B W(1+EB)^2\left(1-\frac{B}{K_B}\right) - M_B B + D_B\nabla^2 B - G(B)H, \tag{1a}$$

$$\partial_t W = P - \frac{NW}{1+RB/K_B} - \Gamma B W(1+EB)^2 + D_W\nabla^2 W, \tag{1b}$$

$$\partial_t H = -M_H H + AG(B)H\left(1-\frac{H}{K_H}\right) - \nabla\cdot\mathbf{J}_H, \tag{1c}$$

over bounded intervals (1D) or bounded rectangles (2D) with periodic boundary conditions. Vegetation growth depends on local water availability and is enhanced by root development, modeled through the nonlinear term $\Lambda BW(1+EB)^2$. Growth is also limited by saturation effects and competition, captured by the factor $1-B/K_B$. Vegetation biomass decreases due to natural mortality at rate M_B and through consumption by herbivores, where the consumption rate $G(B)$ follows a saturating function:

$$G(B) = \frac{\alpha B}{\beta + B},$$

with α representing the maximum per capita consumption rate and β the biomass at which half-maximum consumption occurs.

The soil water balance includes a constant precipitation input P, reduced by evaporation, which is itself mitigated by shading from vegetation ($NW/(1+RB/K_B)$). Plants extract water from the

soil via a nonlinear uptake term proportional to $BW(1+EB)^2$, while water diffuses laterally at a rate D_W.

Herbivores grow by consuming vegetation, reproducing at a rate proportional to the product $G(B)H$, moderated by density dependence via the factor $1 - H/K_H$. Herbivore losses are due to natural mortality at rate M_H and spatial redistribution. Movement is governed by the flux term:

$$\mathbf{J}_H = -D_R(B)\nabla H + HD_V(B)\nabla B,$$

where $D_R(B)$ describes biomass-dependent random movement:

$$D_R(B) = D_{HH}\frac{H\xi^2}{\xi^2 + B^2},$$

and $D_V(B)$ describes biased movement up vegetation gradients (vegetaxis):

$$D_V(B) = D_{HB}\frac{B}{\kappa + B}.$$

In this framework, herbivores tend to move more rapidly in bare-soil regions and slow down as they approach vegetation, while also being drawn toward denser patches. This behavior mimics an exploitation strategy, allowing herbivores to efficiently locate and graze on vegetation.

This system of equations can give rise to a rich variety of spatial and temporal patterns, including stationary vegetation stripes, localized herbivore aggregations, and traveling vegetation–herbivore waves. These emergent behaviors are critical for understanding the resilience of dryland ecosystems under increasing stress due to climate change and population-driven grazing pressure.

2 Direct numerical simulation (DNS)

2.1 Basic setup

We use the python `pypde` package for the results related to the direct numerical simulations (DNS) in Figures 8, 11, 12, 13 in [SUM25]. We recommend the official webpage for this package for step by step instruction for installing it, including its dependencies, and to also go through the example problems (especially the construction of the custom PDE class) given there, and to run these examples to get familiar with the way `pypde` works.

The folder `pydns` contains the necessary files (see Table 1 for an overview). In the following we

Table 1: Scripts and functions in `pydns/`

file	purpose, remarks
`DNSpypde1D2D.py`	used for DNS in one dimensional domain in Figs. 8, 11, and in two dimensional domain in Fig. 13 in [SUM25]
`DNSpypde2D.py`	used for DNS two dimensional domain for Fig. 12 in [SUM25]
`DNS_plot.py`	post processing for h5py data files
`BW.mp4, BH.mp4`	animation of evolution of B, W, H traveling wave
`out_data_t30.h5py`	example output data from `DNSpypde1D2D.py`
`DNSdedalus.py`	`dedalus` implementation of the model

comment on particular steps of the implementation, including some listings; the full `*.py` files contain many further comments. In `pydns/DNSpypde1D2D.py`, the class `BWH` contains all the details of the model. We keep the parameters of the system as global variable in the beginning (see Listing 1).

```
28 #defining the PDE class object for the BWH system containing the paramters
29 #B = vegetation, W = soil water, H = herbivore
30 class BWH(PDEBase):
```

```
    def __init__(self, P = 120, bc = "auto_periodic_neumann"):
        super().__init__
        self.lamda = 0.5            #vegetation growth rate
        self.E = 10.0               #root to shoot ratio
        self.K = 0.9                #maximum vegetation per unit area
        self.M = 11.4               #vegetation mortality rate
        self.DB = 1.2

        self.N = 20.0               #evaporation rate
        self.R = 0.01               #reduced evaporation due to vegetation
        self.GamW = 10.0            #soil water uptake rate
        self.DW = 150.0             #lateral soil water diffusion coefficient

        self.mH = 0.06              #herbivore mortality rate
        self.GamH = 0.3             #herbivore prolifereation rate
        self.AlphaH = 0.6           #vegetation consumption rate
        self.BetaH = 0.82           #satiation biomass
        self.K_H = 150.0            #maximum herbivore per unit area

        self.DHH = 400              #diff. coeff. for random motility
        self.Zeta = 0.001           #reference biomass for 50% random motility drop
        self.DHB = 700              #diff. coeff. for vegetaxis motility
        self.k = 0.0001             #reference biomass for 50% vegetaxis motitity drop

        self.P = P                  #precipitation
        self.bc = bc                #boundary condition
```

Listing 1: `pydns/DNSpypde1D2D.py` parameters of the model.

The initial condition is specified in the `get_initial_state` function (Listing 2).

```
    #definition of the function for IC
    #if a special IC is used it can be input through Bini, Wini, Hini during call
    #def get_initial_state(self, grid, storage, Bini, Wini, Hini):

    #if random IC is used then no need input of Bini, Wini, Hini
    def get_initial_state(self, grid, storage, Lx, mesh):
        # Random IC
        B = ScalarField.random_uniform(grid); W = ScalarField.random_uniform(grid)
        H = ScalarField.random_uniform(grid)

        #if IC is input during call, modify B, W, H from Bini, Wini, Hini respectively
        #one can also modify the random IC by accessing B, W, H data as below
        #B.data = Bini; W.data = Wini; H.data = Hini

        #example of changing IC
        x = np.linspace(0, Lx, mesh)
        B.data = 0.5 + (0.01 * np.sin(2 * np.pi * 4 * x/Lx))
        W.data = 1.2; H.data = 0.25

        #this labels are useful when output data file is analysed later
        B.label = "Plants"; W.label = "Water"; H.label = "Herbivore"
        return FieldCollection([B, W, H])
```

Listing 2: `pydns/DNSpypde1D2D.py` initial conditions.

The rate equation is specified in the `evolution_rate` function (Listing 3).

```
    #rate equations for B, W, H
    def evolution_rate(self, state, t = 0):
        B, W, H = state
        #rate equations implemented in one dimension
        B_t = (self.lamda * B * W * ((1.0 + (self.E * B))**2) * (1.0 - (B/self.K))) - (
    self.M * B) - (self.AlphaH * H * B/(self.BetaH + B)) + (self.DB * B.laplace(self.bc)
    )
```

```
87        W_t = self.P - (self.N * W/(1.0 + (self.R * B/self.K))) - (self.GamW * B * W *
      (1.0 + (self.E * B))**2) + (self.DW * W.laplace(self.bc))
88        H_t = (-self.mH * H) + (self.GamH * self.AlphaH * H * B * (1.0 - (H/self.K_H))/(
      self.BetaH + B)) - ((self.DHB * self.k/(self.k + B)) * ((H.gradient(self.bc) * B.
      gradient(self.bc)))) + ((self.DHB * H * self.k/(self.k + B)**2) * B.gradient(self.bc
      )**2) - ((self.DHB * H * self.k/(self.k + B)) * B.laplace(self.bc)) + ((self.DHH *
      self.Zeta**2/(self.Zeta**2 + B**2)) * H.laplace(self.bc)) - ((self.DHH * 2 * self.
      Zeta**2 * B/(self.Zeta**2 + B**2)**2) * ((H.gradient(self.bc) * B.gradient(self.bc))
      ))
```

Listing 3: `pydns/DNSpypde1D2D.py` rate equations (1D).

We also add a `numba` implementation of the model so that it can be run in parallel where the domain in divided in multiple cores (see source). We recommend relevant webpage for py-pde with `mpi`.

The length of the domain, mesh size and grid construction with periodic boundary condition is given in Listing 4.

```
162 Lx = 4*np.pi; mesh = 1024
163 #defining the grid with periodic boundary condition in one dimension
164 grid = CartesianGrid([[0,Lx]], [mesh], periodic = True)
```

Listing 4: `pydns/DNSpypde1D2D.py` domain and grid (1D)

We then enter value(s) of precipitation in a list. For the value(s), P, the PDE class is initialized. We use a Runge-Kutta scheme for temporal evolution. However a simpler time stepper, the Euler scheme can also be used by changing `scheme` to `euler` inside the `ExplicitMPISolver` function (see source).

The DNS code generates data files in `h5py`. We have provided a `jupyter notebook`, `DNS_plot.ipynb` with methods for postprocessing. With the current parameter values given in the listings, we can obtain the patterned solution at $P = 120mm/y$. We have also provided the "out_data.h5py" and animations, `BW.mp4` and `BH.mp4` obtained using the plot commands in `DNS_plot.ipynb`.

2.2 Instructions for generating the DNS figures in the manuscript

Figure 8 in [SUM25] is obtained from 1D simulation where the initial condition is homogeneous with an added perturbation. The steps to obtain Fig.11 (1D) and Fig.12 (2D) in [SUM25] are similar. For Fig.11, at first the precipitation range is identified where a single patch of vegetation spot is stable. The conventional way to do it is to first obtain a patterned solution in the $B - W$ system without H. The code `DNSpypde1D2D.py` can be easily changed to a simulate $B - W$ model by commenting out the H equation and removing the consumption term from the B rate equations. From the output data file, a domain length of size of a wave length containing one wave is chosen and kept as it is while the data for the rest of the domain is changed to zero. This modified data file is used as initial condition for $B - W$ system with a slightly lower P. This process is repeated until the single patch initial condition is stable. During this process we reduce P by a value of 2 mm/y in each trial step until the desired solution of stable patch of B is obtained. The single spot solution is used as an initial condition for the full $B - W - H$ system with initial condition for H given as a Gaussian function. We have provided all the necessary parts to perform this task in the code as comments. Figure 13 of [SUM25] is obtained in similar step. In the same way as before we isolate a spot in the $B - W$ system and identify a P value where only this spot is stable. We then introduce H through a Gaussian function for the complete $B - W - H$ where the stable single spot solution was used as an initial condition for B and W. The steps to obtain Figure 12 of [SUM25] are simpler. We provide `DNSpypde2D.py` in the same folder where a 2D implementation is provided for convenience. The model is simply run from a homogeneous initial condition with an added perturbation of random noise to sufficiently large t when the desired (stable) solutions emerge. In order to run the system within reasonable time the user advised to use a cluster server where multiple cores are available. The `pyplot` commands for postprocessing are available from the corresponding author of [SUM25] upon request.

The system can also be simulated using the `dedalus` package in python, but we have only used that initially in order to verify a few results related to the bifurcation diagrams, and not for the figures in [SUM25], because `pypde` was faster. Nevertheless, we also provide a basic `dedalus` implementation in `DNSdedalus.py`.

3 Numerical continuation

The folder `bwhcont` contains the files to reproduce the bifurcation diagrams (BDs) presented in the manuscript (see Table 1 for an overview). The 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. We recommend the demos and tutorials at [pde25] to become familiar with the `pde2path` command structure and workflow, see also [Uec21].

Table 2: Scripts and functions in `bwhcont/`

file	**purpose, remarks**
`cmds1, cmds2, cmds3`	Basic BDs for "Parameter set A", with $\alpha = 0.48, \alpha = 0.57$ and $\alpha = 0.603$, respectively, including plotting of BDs, which however in [SUM25] was done via `pyplot`. See `cmds4.m` for an example of "Parameter set B".
`cmdsbpc`	branch, fold and Hopf point continuation
`cmdsdatext`	postprocessing (data extraction for `pyplot`).
`setApar, setBpar`	parameters (sets A and B, respectively), except α_H, which is set in `cmds*.m`
`bwhinit`	Initialization of problem struct p with the required parameter values; generates the FEM matrices by calling `oosetfemops`; sets the `pde2path` parameters according the requirement of the problem
`oosetfemops`	creates the mass matrix M and the Laplacian K for periodic boundary condition
`nodalf`	kinetic terms without the spatial derivatives
`sG`	right hand side of the PDE with the spatial terms in the moving co-ordinate system
`hobrax`	mod of library function hobra
`twswibrax`	mod of library function twswibra
`qf1, qf1der`	standard phase condition for translational invariance in 'x'

3.1 Basic setup

The function `bwinit.m` (Listing 5) initializes the problem `p` with the inputs the domain size `lx`, the number of discretization points `nx`, and the model parameters `par`.[1]

Remark 1 A special parameter in problems such as (1) is the *comoving frame speed* s, which in our setup is $s =$ `par(20)`. This is used in two ways, see e.g., [RU17]: for the continuation of steady states, it serves as a dummy parameter for phase conditions of the form $q(u) = \langle \partial_x u_0, u\rangle$, where u_0 is a chosen profile (usually from the previous continuation step). This is needed for nonhomogeneous steady states due to the translational invariance from the periodic boundary conditions, but s should stay 0 in this case (i.e., numerically 10^{-6}, say). On the other hand, we compute *traveling waves* (TWs) as *relative equilibria* in a frame comoving with the wave speed s by adding $s\partial_x u$ to the rhs of the

[1] For simplicity and reuse, we define the parameter sets A and B in `setApar` and `setBpar`. In `pde2path`, the "solution" vector `u` always holds the fields (here B, W, H) and the parameters, as a long column vector, i.e., `u=[B;W;H;pars]`. The m *active parameters*, i.e., those that are not fixed, are chosen by setting `p.nc.ilam=[i1, i2, ..., im]`.

model, i.e., $s\partial_x(B, W, H)$ to the rhs of (1) in our case. In this case s is exactly the speed of the TW, and is a typical "secondary active" parameter, besides the primary active continuation parameter.

For the handling and setup (via `box2per`) of the periodic boundary conditions we again refer to [RU17], or [Uec21, §4.3] or [DU17]. ⌋

In any case, the first step in `bwinit` is to call `p = stanparam` to generate a problem structure with standard values of numerical parameters and control switches, and to subsequently overwrite some of these with values appropriate for the particular problem. For example we have modified (the library branch–ouput function) `hobra.m` to `hobrax.m` and link it as `p.fuha.outfu=@hobrax`. The function `oosetfemops.m` (Listing 6) assembles the necessary finite element matrices such as stiffness matrix in `p.mat.K` and the mass matrix `p.mat.M` appropriate for periodic boundary condition. The function `sG` (Listing 7) implements the rhs of (1) (actually the negative rhs, because `pde2path` assumes systems of the form $\partial_t u = -G(u)$), calling `nodalf` (Listing 8) to compute the terms without spatial derivatives, and using some preassembled matrices from `p.mat` for the linear diffusion (and advection) terms, but also directly assembles the matrices for the nonlinear diffusion terms.

```
function p=bwinit(lx,nx,par,varargin)
p=stanparam;  % initialize fields with defaults, then overwrite some:
p.nc.neq=3; p.sw.sfem=-1;
p.fuha.outfu=@hobrax; % mod of llibrary function hobra
p.pdeo=stanpdeo1D(lx,2*lx/nx); p.vol=2*lx; % standard 1D PDE object
p.np=p.pdeo.grid.nPoints; p.nu=p.np*p.nc.neq; % # PDE unknowns
p.nc.neig=30; % #eigenvalues to compute, and reference point for this
p.sol.xi=1/p.nu; p.sw.para=2;
p.nc.ilam=6;  % use par(6) (P) for continuation
p.sol.ds=0.1; p.nc.dsmax=10; % initial and maximal stepsize
p.nc.lammin=0.0001; p.nc.lammax=500; % min and max values for cont parameter
p.nc.dlammax=10; %max step size in lam (bifurcation param)
p.file.smod=100; % save each smod'th cont.step, large at init to save few data points,
  % will be decreased for more interesting (patterned) branches
p.sw.bifcheck=2; % bif.detection based on eigenvalues (not LU-decomp and det.)
p.nc.tol=1e-6; % tolerance
p.sw.jac=0;    % switch for jacbian, zero as we use numerical jacobian (no sGjac)
p.nc.mu1=1;    % tolerance for entering BP localization by bisection
%default initial condition
np=p.np; B=zeros(np,1); W=(par(6)/par(7))*ones(np,1); H=zeros(np,1);
p.u=[B; W; H; par']; % initial solution (bare soil) and parameters
p=box2per(p,1);      % switch on periodic BCs, this also calls oosetfemops
p.plot.pcmp=[1 2 3]; p.plot.cl={'k','b','r'}; % plotting settings
screenlayout(p); p.sw.verb=2; % place windows; choose verbosity of output
```

Listing 5: `/bwhcont/bwinit.m`

```
function p=oosetfemops(p)
gr=p.pdeo.grid; fem=p.pdeo.fem;
[K,M,~]=fem.assema(gr,1,1,1); % (scalar) stiffness and mass matrices
M=kron([[1,0,0];[0,1,0];[0,0,1]],M); % system mass matrix
p.mat.M0=p.mat.fill'*M; % adapting to pBCs, here for nonlinearity
p.mat.M=filltrafo(p, M); p.mat.K=filltrafo(p, K); % adapting to pBCs (general)
Kx=fem.convection(gr,1); % advection (scalar), now transform to system and pBCs
Kx=kron([[1,0,0];[0,1,0];[0,0,1]],Kx); p.mat.Kx=filltrafo(p,Kx);
```

Listing 6: `/bwhcont/oosetfemops.m`

```
function r=sG(p,u) % (negative) rhs for BWH model
f=nodalf(p,u); F=p.mat.M0*f; % terms without spatial derivatives
np=p.np; par=u(p.nu+1:end);  % parameters
up=u(1:p.nu); uf=p.mat.fill*up; B=uf(1:np); H=uf((2*np)+1:3*np); % fields
dB=par(5); dW=par(10); dhh=par(15);
zeta=par(16); zeta2=zeta^2; kap=par(18); dhb=par(17);
c1=-dhb*kap*H./(kap+B); c2=dhh*zeta2./(zeta2+B.^2); % diffusion tensors
[K31,~,~]=p.pdeo.fem.assema(p.pdeo.grid,c1,0,0); % nonlin.diffusions
```

```
[K33,~,~]=p.pdeo.fem.assema(p.pdeo.grid,c2,0,0);
K31=filltrafo(p,K31); K33=filltrafo(p,K33);
K=p.mat.K; % preassembled scalar (linear) diffusion
K=[dB*K 0*K 0*K; 0*K dW*K 0*K;  K31 0*K K33];
s=par(20); r=K*up-F-s*p.mat.Kx*up; % comoving frame speed, and full rhs
```

Listing 7: `/bwhcont/sG.m`, rhs of (1), see `nodalf` for terms without spatial derivatives

```
function f=nodalf(p,u) % BWH model, terms without spatial derivatives
par=u(p.nu+1:end); up=u(1:p.nu); % split into pars and u
uf=p.mat.fill*up; % fill to full domain
n=p.np; B=uf(1:n); W=uf(n+1:2*n); H=uf(2*n+1:3*n); % the 3 fields B, W and H
Lam=par(1); E=par(2); K=par(3); M=par(4); pp=par(6); N=par(7); R=par(8);
GamW=par(9); mH=par(11); GamH=par(12); alH=par(13); bH=par(14); KH=par(19);
f1=Lam*B.*W.*(1+E*B).^2.*(1-B/K)-M*B-alH*B.*H./(bH+B);
f2=pp-N*W./(1+R*B/K)-GamW*B.*W.*(1+E*B).^2;
f3=-mH*H+GamH*alH*B.*H.*(1-(H/KH))./(bH+B);
f=[f1; f2; f3];        % the rhs
```

Listing 8: `/bwhcont/nodalf.m`, rhs of (1), terms without spatial derivatives

3.2 Basic bifurcation diagrams

We start explaining the basic steps to compute bifurcation diagrams (BD) based on `cmds1.m`, which computes Fig.3a of [SUM25], i.e., for parameter set A with $\alpha = 0.48$. The other BDs (for $\alpha = 0.57$, `cmds2.m`; for $\alpha = 0.603$, `cmds3.m`; and for parameter set B) work exactly the same way, but contain a few more branches, and hence we also give code snippets how to compute these. After initialization we first continue the "Bare Soil" (BS) branch, and then switch to the "Uniform Vegetation" (UV) branch, see Listing 9.

```
%% driver script to compute BD for "Set A" with alpha=0.48; run cell-by-cell
close all; keep pphome; rng(42);
%% specification of the system, parameter values, domain size, grid size
setApar;  % collect parameters from set A, except AlphaH, which we set now:
AlphaH=0.48;   % maximum rate of plant consumption per unit herbivore
% parameters are refered to by their number in p.nc.ilam; note numbers here!
par=[Lambda,E,K,M,DB,P,N,R,GamW,DW,mH,GamH,AlphaH,BetaH,DHH,Zeta,DHB,k,K_H,s];
%       1            6                 12                          17
nx=256; lx=pi/2;
p=bwinit(lx,nx,par); p=setfn(p,'bwh1/BS');% init and initial folder-name
% Bare soil solution branch (BS)
p=cont(p,1000);     %continuation with number of steps
%% branch switching to uniform vegetation solution branch without herbivore (UV)
% arguments: input dir, branch point no, output dir, initial stepsize
p=swibra('bwh1/BS','bpt1','bwh1/UV',-0.05); p.nc.dsmax=1.0; p=cont(p,100000);
```

Listing 9: `/bwhcont/cmds1.m`; initialization, continuation of bare soil branch, and branch-switching to and continuation of `UV` branch.

From the `UV` branch we switch to the "Uniform (veg. and) Herbivores" (UH) branch, Listing 10.[2]

```
%% Uniform solution branch with vegetation and herbivore (UH) using swibra command
p=swibra('bwh1/UV','bpt3','bwh1/UH',-0.01); p.nc.dsmax=5.0; p=cont(p,10000);
```

Listing 10: `/bwhcont/cmds1.m` continued; swibra to UH;

At low precipitation, UV has a Turing bifurcation point. Using `swibra` the "spatial patterns no herbivores" SP_0 branch is obtained (Listing 11). This is where we first need the phase condition, cf.Remark 1.

[2]This branch switching can fail if the branch point is not properly localized, which can happen if we run the continuation with too large stepsize `ds`, which in turn is sometimes useful to get a quick first overview. For this case, and for completeness, in lines 18-23 of `cmds1.m` we also provide code to "manually switch" to the `UH` branch.

```
%% stationary periodic solution branch of vegetation without herbivore (SP0)
p=swibra('bwh1/UV','bpt2','bwh1/SP0',0.01); pause; % pause to see kernel (bif.direction)
p.sw.bifcheck=0; p.nc.dsmax=0.1; p=cont(p,20);pause; % some steps without phase cond.
p.nc.nq=1; p.nc.ilam=[6,20]; %phase condition with speed s as secondary parameter
p.fuha.qf=@qf1; p.fuha.qfder=@qf1der; %standard phase condition and derivative
p.file.smod=20; p=cont(p,5000);
```

Listing 11: `/bwhcont/cmds1.m` continued; swibra to SP_0;

From the plot commands at the end of `cmds1.m`, we now obtain the BD in Fig.1.

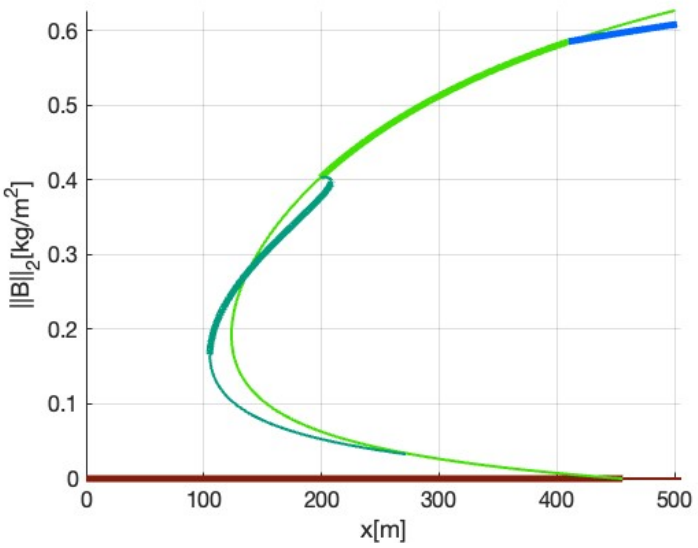

Figure 1: Bifurcation diagram for Parameters set A and $\alpha = 0.48$. This corresponds to Fig.3a of [SUM25].

For $\alpha = 0.57$ and $\alpha = 0.603$, we get the analogs of the above four solution branches, but SP_0 looses stability at low P, yielding a SP_H solution branch (steady patterns with herbivores). This SP_H branch quickly looses stability in a drift bifurcation to a TW branch TW_W. The same happens for $\alpha = 0.603$, and additionally UH loses stability in a Hopf bifurcation at large P to a TW branch TW_H. For simplicity we thus give the commands to obtain SP_H, TW_W and TW_H as snippets from `cmds3.m` for $\alpha = 0.603$. Fig.2 shows the BDs thus obtained from `cmds2.m` and `cmds3.m`.

```
%% stationary periodic solution branch vegetation and herbivore (SPH)
p=swibra('bwh3/SP0','bpt1','bwh3/SPH',0.1); pause;
p.nc.tol=1e-8; p.nc.dsmax=1.0; p=cont(p,100);
```

Listing 12: `/bwhcont/cmds3.m`; `swibra` to SP_H;

```
%% travelling solution branch of both vegetation and herbivore (TW at low P)
% DRIFT bif., hence swibra, not twswibra
p=swibra('bwh3/SPH','bpt1','bwh3/TW_W',0.01); pause;
p.nc.tol=1e-5;p.nc.dsmax=1.0; p.sw.bifcheck=2; p=cont(p,150);
```

Listing 13: `/bwhcont/cmds3.m` continued; swibra to TW at low P;

```
%% travelling solution branch from UH branch (TW at high P)
kwnr=1.0; aux.z=[1 -1i]; % aux arg z to get on TW branch
hp='hpt1'; outb='bwh3/TW_H'; % Hopf point number;output directory
p=twswibrax('bwh3/UH',hp,20,kwnr,outb,aux); %branch switching from Hopf point to TW
p.sol.ds=0.1; p.nc.dsmax=0.1; % now setting phase condition (PC);
p.u0(1:p.nu)=p.tau(1:p.nu); % choose a reference profile u0, and use u0_x for PC
p.u0=p.u0'; p.u0x=p.mat.Kx*p.u0; plotsolu(p,p.u0x,1,1,1); pause
p.u(1:p.nu)=p.u(1:p.nu)+0.01*p.tau(1:p.nu); % set predictor by hand
p.nc.tol=1e-6; p.nc.nq=1; p.nc.ilam=[6;20]; p.fuha.qf=@qf1; p.fuha.qfder=@qjac;  pause
```

```
47 p.nc.dsmax=0.01; p=cont(p,20); par=getaux(p); par(20)    % to check speed
```

Listing 14: `/bwhcont/cmds3.m` continued; twswibrax to the TW branch at high P;

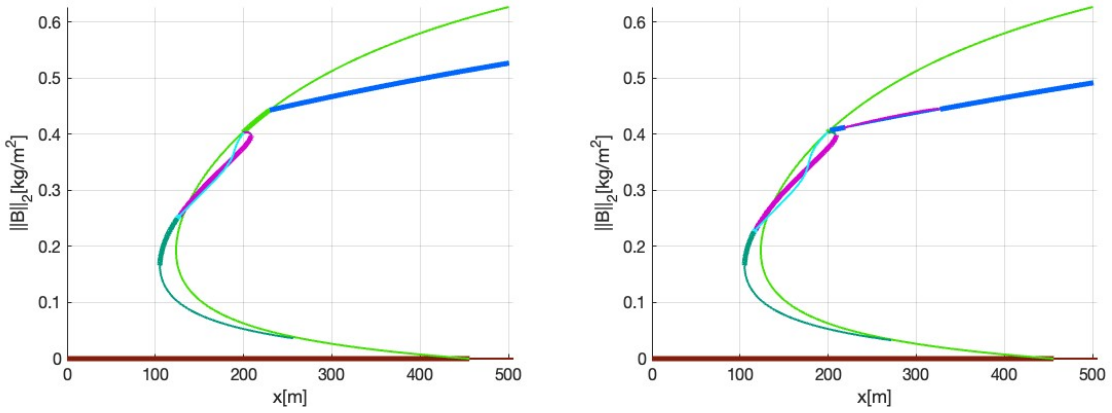

Figure 2: Bifurcation diagram obtained at $\alpha = 0.57$ and $\alpha = 0.603$ for set A. (Fig.3b and Fig.3c of [SUM25]).

The bifurcation diagrams in [SUM25] (Figs.3,5,6,7,14,16) were obtained using scripts as above; see for instance `cmds4.m` for the first BD for "Parameter set B" [SUM25, Fig.6a]. To generate Fig.16, low values of `p.sol.ds` and `p.nc.dsmin` on the order of 10^{-4} were needed to zoom in on a narrow range of P. For a few customization advantage we use `pyplot` (script not provided but available upon request) after extracting the data in a text file. As long as we are only interested in data computed during continuation (here in the custom functions `hobrax` (for steady states) and `hobratwx` (for TWs), linked as `p.fuha.outfu=@hobrax` resp.`p.fuha.outfu=@hobratwx`) we can simply load the last point `p` in a branch-directory and extract the data from `p.branch`, see `cmdsdatext.m`. However, if a posteriori we decide to plot branches of "other data", then we must loop through branches, load points, and compute and store the data, see continuation of `cmdsdatext.m`. Similarly, the solution profiles in Fig.3, 5, 6, 7 are also obtained by data extraction and `pyplot`. Figures 10.b, c were obtained by changing `p.nc.ilam` corresponding to the number of position of D_{HB} with other corresponding parameters and changing the output component in `p.plot.bpcmp`. Figure 15 is also obtained simply by changing `p.plot.bpcmp` for the TW branches for the corresponding value of the parameters.

```
1  %% sample for data extraction for postprocessing (e.g., pyplot) from p.branch
2  % branch=[count; type; ineg; lam; err; ||u_1||_2; userdata]
3  %          1                                     7 .....
4  % here userdata=output of hobrax=[par; m1; m2; m3; m4;  ...; m9],
5  % and m4=normalized-L2 of u1=B is what we want. Since length(par)=20, the
6  % right index of m1 is 6+20+4=30
7  % (In plotbra, the first 6 are not counted, hence there m4 is index c=24).
8  p=loadp('bwh3/BS'); % loads last point from dir
9  bra=p.branch; ineg=bra(3,:); lamv=bra(4,:); l2B=bra(30,:);
10 dat=[lamv', l2B', ineg'];
11 ptr=fopen('BS.txt','w');         %change output directory accordingly
12 fprintf(ptr,'%e\t%e\t%e\n',P');   fclose(ptr);
```

Listing 15: `/bwhcont/cmdsdatext.m` bifurcation diagram data extraction.

3.3 Phase diagrams: branch point continuation

The phase diagrams in Figs.9.a and 10.a in [SUM25] were partly obtained using branch point continuation (BPC), fold point continuation (FPC) and Hopf point continuation (HPC), see [Uec21, §3.6.1]. However, because we refrain from analytical Jacobians, here we advise caution for these continuations whenever H is non-zero or branch points are close together. In these cases, `p.nc.tol` often needs to be adjusted during continuation. Whenever there are convergence error the boundaries between these

stable solutions are completed manually from the bifurcation diagram. Moreover, we checked the boundaries of the phase diagrams using direct numerical simulation. Altogether, this results in long and complicated scripts, and therefore in `cmdsbpc.m` we only explain the basic setup of BPC, FPC and HPC.

Listing 16 shows the commands for branch point continuation for branch point on UV which generates SP_0. Examples of FPC and HPC are given subsequently.

```
%% Branch point continuation (BPC) in par(13)=AlphaH of the Turing BP on UV
p=bpcontini('bwh3/UV','bpt3',13,'bwh3/SP0boundary'); plotsol(p); % init and plot
p.nc.dlammax=0.01;p.nc.lammax=0.608;p.sw.bifcheck=0; p.nc.lammax=35; % HU
p.sw.foldcheck=0; %switch off fold detection
p.sol.ds=0.001; p.nc.tol=1e-7; p.nc.del=0.001; p.nc.njthreshsp=1e4;
p.sw.spjac=0; % numjac for jac of extended system
p.nc.dsmax=0.01; p.nc.dsmin=0.00001;
p.file.smod=1; mclf(2); p=cont(p,20);
%% plotting
f=3; mclf(f); c=6; plotbra('bwh3/SP0boundary',f,c,'cl',p2pc('g1'));
xlabel('\alpha'); ylabel('P'); grid on; box on;
```

Listing 16: `/bwhcont/cmdsbpc.m`; BPC of Turing BP on UV, and plotting; subsequently FPC and HPC.

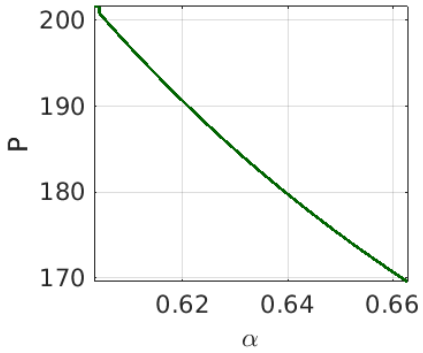

Figure 3: Sample of branch point continuation; α dependence of P for the BP of SP0 from UV.

References

[DU17] T. Dohnal and H. Uecker. Periodic boundary conditions in pde2path, 2017. Available at [pde25].

[pde25] pde2path. `https://pde2path.uol.de/`, 2025.

[RU17] J.D.M. Rademacher and H. Uecker. Symmetries and freezing of 1D problems in pde2path – a tutorial via some Ginzburg-Landau and FHN models, 2017. Available at [pde25].

[SUM25] J. Singha, H. Uecker, and E. Meron. Traveling vegetation–herbivore waves may sustain ecosystems threatened by droughts and population growth, 2025. Preprint.

[Uec21] H. Uecker. *Numerical Continuation and Bifurcation in Nonlinear PDEs*. SIAM, 2021.