Numerical continuation for the spatial model of 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

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 numerical continuation using `pde2path`.

1 Brief description of the model

In [SUM24] we consider a spatially explicit model that describes 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:

$$\begin{aligned}
\partial_t B &= \Lambda BW(1+EB)^2\left(1-\frac{B}{K_B}\right) - M_B B + D_B\nabla^2 B - G(B)H,\\
\partial_t W &= P - \frac{NW}{1+RB/K_B} - \Gamma BW(1+EB)^2 + D_W\nabla^2 W,\\
\partial_t H &= -M_H H + AG(B)H\left(1-\frac{H}{K_H}\right) - \nabla\cdot\mathbf{J}_H.
\end{aligned}$$

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 of the model

We use `pypde` package in python to produce all the results related to the direct numerical simulations (DNS) in Figures 8, 11, 12, 13 in [SUM24]. Before proceeding it is advised that the user should visit the official webpage for this python package for step by step instruction for installing it, all of its dependancies. We advise that the user should go through the example problems, specially the construction of the custom PDE class, given in there and run these examples using the installed package to get familier with the way it works.

The folder `Direct numerical simulation` contains all the necessary files (see Table 1 for an overview).

Table 1: Scripts and functions in `Direct numerical simulation/`

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 [SUM24]
`DNSpypde2D.py`	used for DNS two dimensional domain for Fig. 12 in [SUM24]
`DNS_plot.py`	post processing for h5py data files
`BW.mp4, BH.mp4`	animation of evolution of B, W, H travelling wave
`out_data_t30.h5py`	example output data from `DNSpypde1D2D.py`
`DNSdedalus.py`	`dedalus` implementation of the model

In `Direct numerical simulation/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).

```
31 #defining the PDE class object for the BWH system containing the paramters
32 #B = vegetation, W = soil water, H = herbivore
33 class BWH(PDEBase):
34     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: `Direct numerical simulation/DNSpypde1D2D.py` parameters of the model ;

The initial condition is specified in `get_initial_state` function (see listing 2).

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

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

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

        #example of changing initial condition
        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: `Direct numerical simulation/DNSpypde1D2D.py` initial conditions;

The rate equation is specified in `evolution_rate` function (see listing 3).

```
95      #rate equations for B, W, H
96      def evolution_rate(self, state, t = 0):
97          B, W, H = state
98
99
100         #rate equations implemented in one dimension
101         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)
        )
102         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))
103         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))
        ))
104
105         #rate equations implemented in two dimension
106         #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)
        )
107         #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))
108         #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) @ B.gradient(self.bc))) - ((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))))
109
110         return FieldCollection([B_t, W_t, H_t])
```

Listing 3: `Direct numerical simulation/DNSpypde1D2D.py` rate equations;

. 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 advise the user to visit the relevant webpage for py-pde. The length of the domain, mesh size and grid construction with periodic boundary condition after the PDE class is completely defined (see listing 4).

```
171 #domain size in one dimension
172 #to implement in two dimension, use Ly for domain length in one dimension
173 Lx = 4*np.pi
174 mesh = 1024
175
176 #defining the grid with periodic boundary condition in one dimension
177 grid = CartesianGrid([[0,Lx]], [mesh], periodic = True)
```

Listing 4: `Direct numerical simulation/DNSpypde1D2D.py` domain details and grid;

. We then enter value(s) of precipitation in list. For the value(s), P, the PDE class is initialised. We use 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` (see source). The DNS code generates data files in `h5py`. We have provided a `jupyter notebook`, `DNS_plot.ipynb` where methods for loading reading `h5py` is given. 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`. The user is advised to run the codes as background task or in a cluster server as the iterations are slow.

3 Instructions for generating the DNS figures in the Manuscript

Figure 8 in [SUM24] is obtained from simulation of one dimensional implementation where the initial condition is homogeneous with an added perturbation. The steps to obtain Figure 11 and 12 in [SUM24] are similar with the difference that Fig. 11 is for an one dimensional domain while Fig. 12 is in two dimension. For Figure 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 was stable as it is. 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 used as an initial condition for the full "B, W, H" system. Initial condition for the "H" is given as a Gaussian function. We have provided all the necessary parts to perform this task in the code as comments. Figure 13 of [SUM24] is obtained in similar step as. However here we first obtain the spot pattern in the "B, W" model. In the same way as before we isolate a spot and identify 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 [SUM24] is simpler. We provide `DNSpypde2D.py` in the same folder where a full two dimensional implementation is provided for convenience. The model is simply simulated from a homogeneous initial condition with an added perturbation of random noise to a sufficiently large amount of time when desired solutions become stable. In order to run the system within reasonable time the user advised to use a cluster server where multiple cores are available. The particular pyplot commands to generate these are available from the corresponding author of [SUM24] upon request.

The system can also be simulated using the `dedalus` package in python but we have not used it explicitly to generate the figures in [SUM24]. We have only used it initially in order to verify a few results related to the bifurcation diagrams. In one dimension we could not make a parallel implementation resulting in a slow simulation time. For this reason we did not pursue this package and switched to `pypde` for all DNS needs for [SUM24]. However we have provided a basic `dedalus` implementation in `DNSdedalus.py` in case the user is interested.

4 Numerical continuation

All scripts in `Numerical continuation/bwhcont` 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 folder `bwhcont` contain cmds.m with cell blocks beginning with a brief description in each block. This folder contains all the necessary files to reproduce the bifurcation diagrams presented in the manuscript (see Table 1 for an overview). Before making any modifications, we recommend consulting the relevant documentation or references [Uec21], [Uec25] to become familiar with the `pde2path` command structure and workflow.

The function `bwinit.m` initialises the problem (see listing 5), with the inputs the domain size `lx`, the number of discretization points `nx`, the model parameter `par`, an initial homogeneous solution `[B, W, H]`, and the output folder name `'bwh'`. The basic procedure is to use `p = stanparam` to general a problem structure with standard values of numerical parameters and control switches to obtain the bifurcation branch. The default value of are often overwritten to make it suitable for the problem description, for example we have modified `hobra.m` to `hobrax.m` as function handler for output. The function `oosetfemops.m` creates 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 (see

Table 2: Scripts and functions in `bwhcont/`

file	purpose, remarks
`cmds1`	continuation of homogeneous branches and branch switching to and continuation of Turing branches and travelling wave branches; to cont. speed
`cmds2`	branch, fold and hopf point continuation
`cmdsplot`	plot commands for the bifurcation diagrams
`cmdsdatext`	to extract data to text files
`bwhinit`	Initialisation 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 coordinate system
`hobrax`	mod of library function hobra
`twswibrax`	mod of library function twswibra
`qf, qf1, qf1der, qjac`	standard phase condition for translational invariance in 'x'

listing 6).

```
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.sol.xi=1/p.nu; p.file.smod=1; p.sw.para=2; p.sw.foldcheck=1;
p.nc.ilam=6;  % use par(6) (P) for continuation
p.sol.ds=0.1; p.nc.dsmax=10.0; % 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=20; % save each smod'th cont.step, increase saves less data points
p.sw.bifcheck=2; %
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
p.sw.verb=2;
p.nc.neig=30; % #eigenvalues to compute, and reference point for this
p.nc.eigref=-4;
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); % FEM/mass matrices
M=kron([[1,0,0];[0,1,0];[0,0,1]],M);
p.mat.M0=p.mat.fill'*M;
p.mat.M=filltrafo(p, M);
p.mat.K=filltrafo(p, K);
```

```
Kx=fem.convection(gr,1); Kx=kron([[1,0,0];[0,1,0];[0,0,1]],Kx);
p.mat.Kx=filltrafo(p,Kx);
```

Listing 6: `/bwhcont/oosetfemops.m`

5 Demo bifurcation diagrams

Using the bare soil solution as the initial guess, we first continue first on the homogeneous branch, (see listing 7).

```
%% Please each section of this code to get the branches of
close all; keep pphome;rng(42);
%% specification of the system, parameter values, domain size, grid size
Lambda=0.5;              % rate of vegetation growth per unit soil water
E=10;                    % root to shoot ratio
K=0.9;                    % maximal standing biomass per unit area
M=11.4;                  % plant mortality rate
DB=1.2;                  % seed dispersal rate
P=0.0;                   % precipitation
N=20.0;                  % evaporation rate
R=0.01;                  % Reduction in evaporation due to shading (dimensionless)
GamW=10.0;               % water uptake rate
DW=150;                  % lateral soil water diffusion
mH=0.06;                  % herbivore mortality rate
GamH=0.3;                % fraction of consumed biomass used in herbivore production
AlphaH=0.603;               % maximum rate of plant consumption per unit herbivore
BetaH=0.82;               % satiaton biomass
DHH=400;              % maximal random motility
Zeta=0.001;              % reference biomass at which the motility drops to 50%
DHB=700;               % maximal vegetaxis motility
k=0.0001;                % reference biomass at which the motility drops to 50%
K_H=175;                 % maximum herbivore capacity per unit area
s=0;                     % speed
% 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,'bwh3/BS');% init and initial folder-name
% Bare soil solution branch (BS)
p=cont(p,1000);     %continuation with number of steps
```

Listing 7: `/bwhcont/cmds1.m` parameters and bare soil solutions branch;

From the first branch point of the `BS` branch, we continue the `UV` branch (see Listing 8).

```
%% branch switching to uniform vegetation solution branch without herbivore (UV)
% arguments: input dir, branch point no, output dir, initial stepsize
p=swibra('bwh3/BS','bpt1','bwh3/UV',-0.05); pause; % pause to inspect kernel in Fig.6
p.nc.dsmax=1.0; p=cont(p,100000);
```

Listing 8: `/bwhcont/cmds1.m` continued; swibra to UV;

From the `UV` we then continue to the `UH` branch (see listing 9).

```
%% Uniform solution branch with vegetation and herbivore (UH) using swibra command
p=swibra('bwh3/UV','bpt3','bwh3/UH',-0.01); p.nc.dsmax=5.0;
p.file.smod=20; p=cont(p,10000);
```

Listing 9: `/bwhcont/cmds1.m` continued; swibra to UH;

Here if `swibra` does not work, we can manually switch to the `UH` branch by using `loadp` jump from the branch point to `UH` (see listing 10).

```
%% Uniform solution branch with vegetation and herbivore (UH) through manual
    initialisation if swibra does not work
p=loadp('bwh2/UV','bpt3','bwh2/UH'); %load from a pt in the in dir to an out dir
p.u(p.nu+6)=p.u(p.nu+6)+1.0; %manually increasing the lam
n=p.nu/3; p.u(2*n+1:3*n)=2.0;p=resetc(p); %changing H and resetting struc count
p.sw.bifcheck=2;
p.sol.ds=1.0; p.nc.dsmax=2;
p=cont(p,200);
```

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

At low precipitation, `UV` has a turing bifurcation point. Using `swibra` the SP_0 branch is obtained. Here also use the phase condition (see listing 11).

```
%% stationary periodic solution branch of vegetation without herbivore (SP0)
p=swibra('bwh3/UV','bpt2','bwh3/SP0',0.01); pause;
p.sw.bifcheck=0; p.nc.dsmax=0.1; p=cont(p,20);pause;
p.nc.nq=1; p.nc.ilam=[6,20]; %phase condition on and adding speed as another bifurcation
    parameter
p.fuha.qf=@qf1; p.fuha.qfder=@qf1der; %standard phase condition and derivative
p.sw.qjac=1;    %phase condition jacobian
p.sw.bifcheck=2;  p.nc.dsmax=5.0;p=cont(p,168);
```

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

Using the above settings for $\alpha = 0.48$ and setting the other parameters in set A, we obtain the complete bifurcation diagram (See Fig 1).

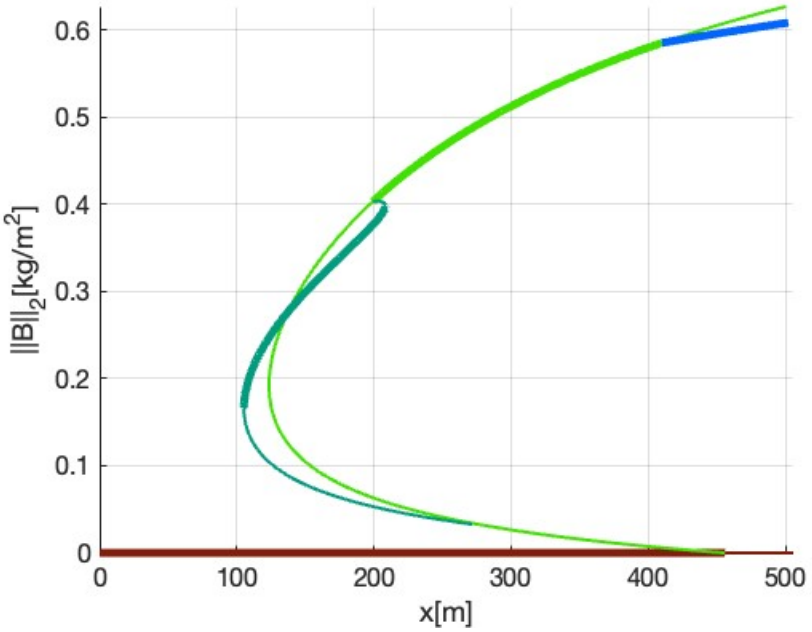

Figure 1: Bifurcation diagram obtained at $\alpha = 0.48$ using `plotbra` for parameters in set A . This is Fig. 3a of [SUM24].

For $\alpha = 0.57$, we get the 4 solution branches as before but SP_0 looses stability at low P. From here we access the SP_H solution branch using `swibra` from the corresponding branch point on SP_0 (see listing 12).

```
%% 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/cmds1.m` continued; swibra to SP_H;

This SP_H branch quickly looses stability and from there, the TW branch can be continued using `swibra` (see listing 13). In this way we can obtain the bifurcation diagram in Fig. 3b of [SUM24] (see figure 2).

```
%% 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/cmds1.m` continued; swibra to TW at low P;

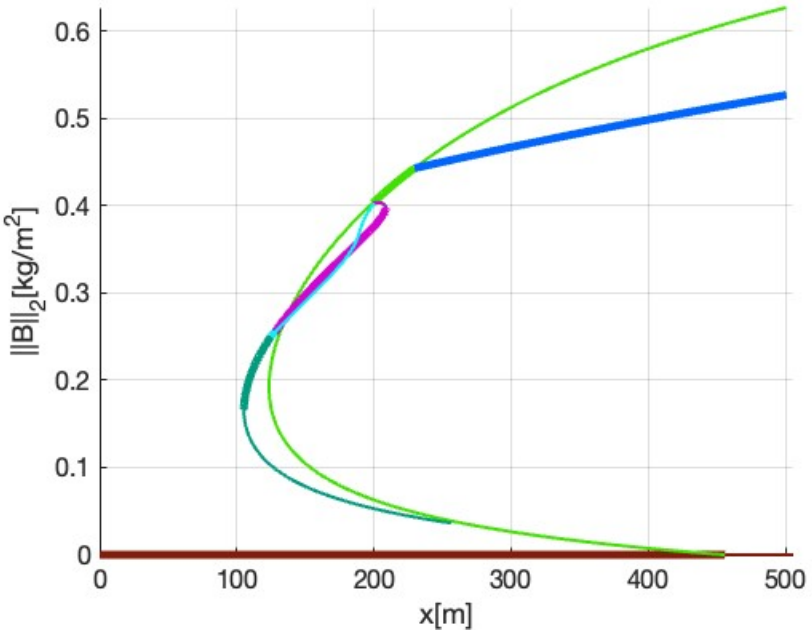

Figure 2: Bifurcation diagram obtained at $\alpha = 0.57$ for set A. (Fig. 3b of [SUM24]).

Setting, $\alpha = 0.603$, we again get all of the 6 solutions branches as before. However in this case UH solution looses stability to in hopf bifurcation. Using `twswibrax` we switch to the travelling wave branch here (see listing 14). In this way we can generate Fig. 4.

```
%% travelling solution branch from UH branch (TW at high P)
kwnr=1.0;      %guess for spatial wave number
aux.z=[1 -1i]; %auxilary arg for twswibrax, this comb of guesses of z1 and z2 works well
     for hp point on UH
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
    travelling wave branch
p.sol.ds=0.1; p.nc.dsmax=0.1;
p.u0(1:p.nu)=p.tau(1:p.nu);     %reference profile
p.u0=p.u0'; p.u0x=p.mat.Kx*p.u0; % setting PC
plotsolu(p,p.u0x,1,3,1);
p.u(1:p.nu)=p.u(1:p.nu)+0.01*p.tau(1:p.nu);
p.nc.tol=1e-6; p.nc.nq=1;
p.nc.ilam=[6;20]; p.fuha.qf=@qf; p.sw.qjac=1; p.fuha.qfder=@qjac;
p.sw.bifcheck=2; pause;
p.nc.dsmax=0.2; p=cont(p,570);
par=getaux(p); par(20)          %to check speed
```

Listing 14: `/bwhcont/cmds1.m` continued; twswibrax to the TW branch at high P;

All of the bifurcation diagrams in [SUM24] (Figs. 3, 5, 6, 7, 14, 16) were obtained using the above for different values of parameters. To generate Fig. 16, low values of `p.sol.ds` and `p.nc.dsmin`

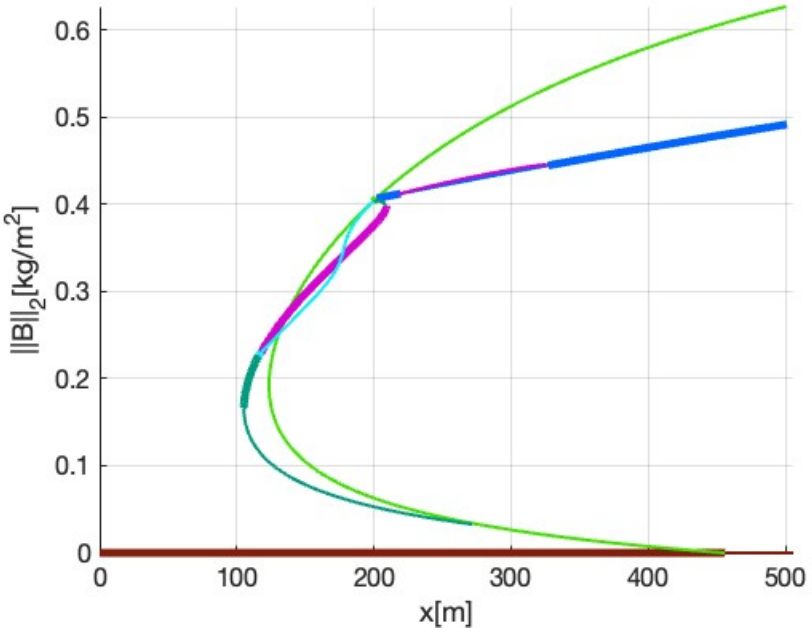

Figure 3: Bifurcation diagram obtained at $\alpha = 0.603$ for set A. (Fig. 3c of [SUM24]).

on the order of 10^{-4} were used, as they were necessary to zoom in on a narrow range of P. For a few customisation advantage we use pyplot (script not provided but available upon request) after extracting the data in a text file (see `cmdsdatext.m`). For any `p.file.smod` a proper post-processing commands to extract from the `MATLAB` is given in listing 15.

```
%% it's all on the branch! (the trick is to find it!); -see also bradat.m
% branch=[count; type; ineg; lam; err; ||u_1||_2; userdata]
%          1                                      7 .....
% here userdata=output of hobrax=[par; m1; m2; m3; m4;  ...; m9],
% and m4=normalized-L2 of u1=B is what we want. Since length(par)=20, the
% right index of m1 is 6+20+4=30
% (In plotbra, the first 6 are not counted, hence there m4 is index c=24).
p=loadp('bwh/BS'); % loads last point from dir
bra=p.branch; ineg=bra(3,:); lamv=bra(4,:); l2B=bra(30,:);
P2=[lamv', l2B', ineg']; % this now contains all data from ALL THE COMPUTED points,
% no matter how many points are saved (no matter what smod is)
% Of course, this needs that you know in advance what you want. If you need
% something which is not computed in p.fuha.outfu (happens to me!), then
% you must do it your way...
%% check for SPH:
npt=8; nski=20; P=zeros(npt,3);         %array size depends on the number of points in
    the dir
for i=1:npt;
    pt=['bwh/SPH/pt' mat2str((i-1)*nski)]; tmp=load(pt);tmpp=tmp.p;
    P(i,1)=tmpp.u(tmpp.nu + 6); np=p.nu/p.nc.neq;n=p.nu;
    P(i,2)=(tmpp.u(1:np)'*(p.mat.M(1:np,1:np)*tmpp.u(1:np)))/tmpp.vol; % (normalized) L2
     of B
    P(i,2)=sqrt(P(i,2));
    P(i,3)=tmpp.sol.ineg;
end
ptr=fopen('SPH.txt','w');                %change output directory accordingly
fprintf(ptr,'%e\t%e\t%e\n',P'); fclose(ptr);
%%
p=loadp('bwh/SPH','pt20'); bra=p.branch; ineg=bra(3,:); lamv=bra(4,:); l2B=bra(30,:);
P2=[lamv', l2B', ineg']; P2
```

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

Similarly the the solution profiles in Fig. 3, 5, 6, 7 are also obtained (see listing 16).

```
%% solution profile data
n = p.nu/3;
p=loadp('bwhcont/BS','pt198');
B = p.u(1:n);
W = p.u(n+1:2*n);
H = p.u(2*n+1:3*n);
ptr=fopen('bwhcont/pt198.txt','w');
fprintf(ptr,'%f\t%f\t%f\n',[B W H]');
fclose(ptr);
```

Listing 16: `/bwhcont/cmdsdatext.m` continued; solution profile data extraction;

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` to observe speed (see listing 17). Figure 15 is also obtained simply by changing `p.plot.bpcmp` for the TW branches for the corresponding value of the parameters.

```
%% continuation wrt DHB
p=loadp('bwh/TW','pt137','bwh/speed_DHB');
p.sol.ds=10.0;p.nc.ilam=[17,20];
p=resetc(p);p.nc.lammax=6000;pause;
p.nc.tol=1e-4;
p.sw.bifcheck=0; p.nc.dsmax=10.0;
p.plot.bpcmp=23;p=cont(p,3); pause;
p.nc.nq=1;  p.fuha.qf=@qf1;
p.fuha.qfder=@qf1der; p.sw.qjac=1; % switch on PC
p.sw.bifcheck=2;p=cont(p,1000);
```

Listing 17: `/bwhcont/cmds1.m` continued; speed vs D_{HB} diagrams;

As before, their data were extracted and plotted in pyplot.

6 Phase diagrams : branch point continuation

The phase diagrams in Figs. 9.a and 10.a in [SUM24] were mostly obtained using branch point continuation method in `pde2path` (see listings 18, 19, 20 for branch, fold and hopf point continuation respectively.). We advise caution for these continuations whenever `H` is non-zero or branch points are nearby as the `p.nc.ntol` needs to adjusted during continuation. Whenever there are convergence error the boundaries between these stable solutions are completed manually from the bifurcation diagram. We also advise that the boundary of the phase diagram should be checked always using direct numerical simulation whether above and below the boundary the desired solutions are obtained.

The listing 18 shows that `pde2path` commands for branch point continuation for branch point on UV which generates UH that runs for few continuation steps (see figure **??**).

Listing 18: `/bwhcont/cmds2.m` continued; branch point continuation;

```
%% For branch point continuation to find variation in the position of the turing bif
p=bpcontini('bwh1/UV','bpt3',13,'bwh1/Turing_UV'); %branch point continuation w.r.t
plotsol(p); mclf(2); pause %solution profile;to clear fig 2;
p.nc.dlammax=0.0001;p.nc.lammax=1.0;p.sw.bifcheck=0;
p.sw.foldcheck=0; %switch of fold
p.sol.ds=-0.01;
p.nc.tol=1e-7; p.nc.del=0.001; p.nc.njthreshsp=1e3;
p.sw.spjac=0; %for fuha.spjac
p.nc.dsmax=0.01; p.nc.dsmin=0.00001;
p.file.smod=1; p=cont(p,10);
```

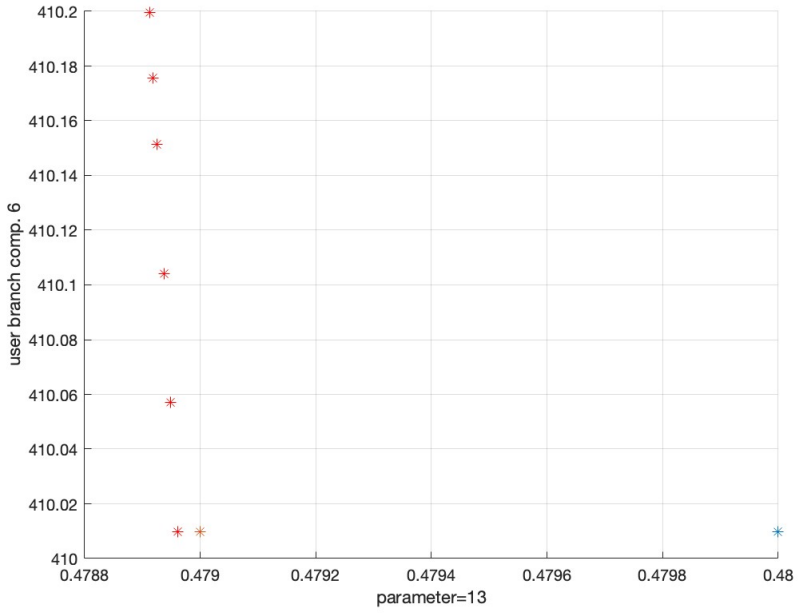

Figure 4: Branch point continuation.

Both the fold point continuation and hopf continuations (see listings 19, 20) runs for few steps initially however we get convergence errors so therefore we mostly use manual evaluation of the bifurcation diagrams and obtain the phase boundaries.

```
%% for fold point continuation
p=spcontini('bwh3/SP0','fpt2',13,'bwh3/SP0_fold_point'); %fold point cont.
huclean(p);
plotsol(p); pause
p.nc.dlammax=10; p.nc.lammax=1.5;
p.nc.del=1e-2;p.nc.njthresh=1e-2; p.nc.njthreshsp=1e5;
p.sol.ds=0.0001; p.plot.bpcmp=p.nc.ilam(2);
p.nc.tol=1e-5; p.sw.spjac=0; %for fuha.spjac
p.file.smod=1; p.sw.bifcheck=0; p.sw.foldcheck=0;
p.sw.verb=2; p.nc.dsmax=0.001; p=cont(p,50);
```

Listing 19: `/bwhcont/cmds2.m` continued; fold point continuation;

```
%% Hopf point continuation
p=hpcontini('bwh3/UH','hpt1',13,'bwh3/UH_hopf_point'); %hopf point cont
huclean(p); plotsol(p); p.nc.dlammax=0.0001;
p.nc.lammax=0.608;pause
p.plot.bpcmp=p.nc.ilam(2);
```

Listing 20: `/bwhcont/cmds2.m` continued; hopf point continuation;

References

[SUM24] J. Singha, H. Uecker, and E. Meron. Traveling vegetation–herbivore waves may sustain ecosystems threatened by droughts and population growth, 2024. Preprint.

[Uec21] Hannes Uecker. Numerical continuation and bifurcation in nonlinear pdes. *SIAM*, 2021.

[Uec25] Hannes Uecker. pde2path - a matlab package for continuation and bifurcation in system of pdes, v3.1. 2025.