MATH 199C: Spring 2024

Sijia Zhang

June 2024

Abstract

A summary of independent study throughout Spring 2024, containing numerical approximation for partial differential equations by finite difference methods and Newton's method.

1 Background

Finding stable states in the ecosystem directly relates to the theory of tipping points, which contributes to our understanding of ecosystem functioning. With increasing stress, an ecosystem persists in one of its stable states until a critical point (also called a tipping point) is reached and a critical transition (or tipping) to another stable state occurs. In the paper, the authors paid attention to a forest-savanna model and tried to find the stable states of the model.

1.1 Model Description

Consider logistic growth, which allows the forest and savanna biomass to increase until their carrying capacity is reached. Let f and s be a function of x and t, representing the biomass of forest and savanna, respectively. Thus, the Forest-Savanna equations can be expressed as:

$$\begin{cases} \frac{\partial f}{\partial t} = \delta \frac{\partial^2 f}{\partial x^2} + \mu f(1-f) - asf - mf \\ \frac{\partial s}{\partial t} = \frac{\partial^2 s}{\partial x^2} + s(1-s) - bfs - ns \end{cases} \tag{1}$$

Here, s and f are the dimensionless state variables and μ, a, b, m, n are the dimensionless parameters of the model.
To simplify the equations, we introduce another two parameters α and β, which satisfies: $\alpha = \mu = m$, $\beta = 1 - n$. Equations (1) then become:

$$\begin{cases} \frac{\partial f}{\partial t} = \delta \frac{\partial^2 f}{\partial x^2} + \alpha f - \mu f^2 - afs \\ \frac{\partial s}{\partial t} = \frac{\partial^2 s}{\partial x^2} + \beta s - s^2 - bfs \end{cases} \tag{2}$$

with parameters $\mu, \delta, a, b, \alpha, \beta$.

2 Newton's Method by Jacobian

2.1 Definition

Newton's method is useful in numerical analysis which is used to find the roots of a function by producing successively better approximations.
The idea is to start with an initial guess x_0, then to approximate the function by its tangent line, and finally to compute the x-intercept of this tangent line. This x-intercept will typically be a better approximation to the original function's root than the first guess, and the method can be iterated.

For a differentiable function f, we take the tangent line to the curve $f(x)$ at the initial guess x_0 which intercepts the x-axis at x_1, then the slope is:

$$f'(x_1) = \frac{f(x_0) - 0}{x_0 - x_1}$$

Thus, we can solve for x_1:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Repeat the iteration, we can obtain a formula for x_{n+1}:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Jacobian of a function is taking the first order derivative of the function, at the respective position, i.e.:

$$Jacobian = \frac{df}{dx}$$

2.2 Solving Newton's Method by Jacobian

Try to find the positive roots of:
1. $x^2 - 3 = 0$
2. $\begin{cases} x^2 - 3 = 0 \\ x^2 + y^2 = 4 \end{cases}$

Idea. To solve these equations by Newton's Method, we use $fsolve$ in MATLAB.
$fsolve$ is the nonlinear system solver that solves a problem specified by $F(x) = 0$ for x, where $F(x)$ is the function that returns a vector value. x is a vector or a matrix. For example, $x = fsolve(function, x_0)$ starts at x_0 and tries to solve the equations for $function(x) = 0$, an array of zeros.
We first need to write them into a function that is equal to 0, i.e. write in the form of $F(x)$ or $F(x, y) = 0$. By solving Newton's method by Jacobian, we define Jacobian at function part, and add "option" at fsolve, with opention define as "Jacabian, on", written as :

$$options = optimset('Jacobian','on');$$

$$fsolve(...,'option')$$

```
Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>

ans =
   1.732050810014728
Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>

ans = 2×1
     1.732050810014728
     1.000000000000000
```

Figure 1: Result we got after solving the function by Newton's method by Jacobian.

3 Solve Forest-Savanna Equations by Traveling Waves

A traveling wave is a solution of a partial differential equation (PDE) that propagates with a constant speed while maintaining its shape in space. Traveling waves are a common phenomenon in biology, evidenced by the fact that Chapter 13 of the authoritative work "Mathematical Biology" by Murray is dedicated to biological waves. In the context of the forest-savanna model, we can express graphs of forest and savanna with the help of traveling waves.
Define a variable $z = x + ct$, where c is the traveling speed of the wave F and S. Thus, we can express F(x,t) and S(x,t) by F(z) and S(z). Define U(z) as an array with length $2Nx + 1$, where $U(z) = [F(z), S(z), c]$.
By applying finite difference method to equations (2), we can express (2) as:

$$\begin{cases} E_F : cD_1F - \delta D_2F + \alpha F - \mu F^2 - aFS = 0 \\ E_S : cD_1S - D_2S + \beta S - S^2 - bFS \end{cases} \tag{3}$$

Since two equations E_F, E_S are not sufficient to solve for three unknown variablesF, S, c. We are going to introduce another equation E_C:

$$E_C : \begin{bmatrix} D_1F \\ D_2S \end{bmatrix}^T \cdot \left(\begin{bmatrix} F \\ S \end{bmatrix} - \begin{bmatrix} F_0 \\ S_0 \end{bmatrix} \right) = 0 \tag{4}$$

Introduce a new Matlab command "fsolve". fsolve uses the idea of Newton's method and attempts to solve a system of equations by minimizing the sum of squares of the components.
Solves a problem specified by F(x) = 0 for x, where F(x) is a function that returns a vector value and x is a vector or a matrix. Acting as a nonlinear system solver, fsolve can be used in the form:

$$x = fsolve(fun, x_0, options)$$

In order to make the code run faster, we introduce Jacobian to reduce computations. Jacobian can be expressed as:

$$Jacobian = \begin{pmatrix} D_FE_F & D_SE_F & D_CE_F \\ D_FE_S & D_SE_S & D_CE_S \\ D_FE_C & D_SE_C & D_CE_C \end{pmatrix}$$

where D_F, D_S, D_C represents derivatives over W, N, C, respectively.
Plug in equations (5) and (6), Jacobian is a $2N + 1$ by $2N + 1$ matrix:
$D_FE_F = cD_1 - \delta D_2 + \alpha - 2\mu spdiags(F, 0, Nx, Nx) - aspdiags(S, 0, Nx, Nx)$
$D_SE_F = aspdiags(F, 0, Nx, Nx)$
$D_CE_F = D_1F$
$D_FE_S = -bspdiags(S, 0, Nx, Nx)$
$D_SE_S = cD_1 - D_2 + \beta - 2spdiags(S, 0, Nx, Nx) - bspdiags(F, 0, Nx, Nx)$
$D_CE_S = D_1S$
$D_FE_C = (D_1F)^T + (F - F_0)^T D_1$
$D_SE_C = (D_1S)^T + (S - S_0)^T D_1$
$D_CE_C = 0$
where *spdiags* is a Matlab command used to extract nonzero diagonals and create sparse band and diagonal matrices.
Using the idea of Newton's method by applying *fsolve*, we can generate the solution for Forest and Savanna.

3.1 tanh-like Inital Condition

With parameters $a = 1.3, b = 2, m = 0.02, n = 0.4, \mu = 0.1, \delta = 0.0001$, and tanh-like initial condition: $c_0 = 2, F_0 = 0.8 * (0.5 + 0.5 * tanh((x - L/2))), S_0 = 0.6 * (0.5 - 0.5 * tanh((x - L/2)))$, we can solve the Forest-Savanna model (Figure 2, Figure 3, and Figure 4):

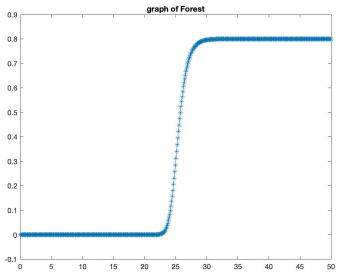

Figure 2: Traveling wave of forest. From the graph, we can tell that the forest grows from F = 0 and finally reaches a steady state at F = 0.8.

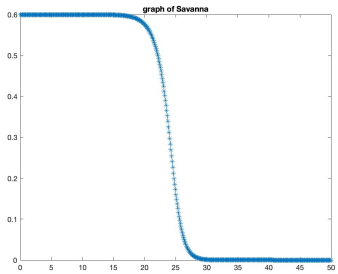

Figure 3: Traveling wave of savanna. From the graph, with parameters a = 1.3; b = 2; m = 0.02; n = 0.4; mu = 0.1, we can tell that the savanna grows from S = 0.6 and finally reaches a steady state S = 0.

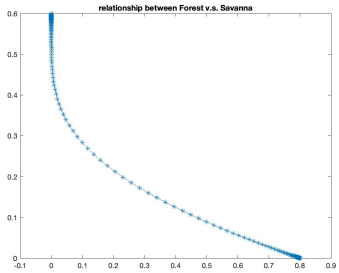

Figure 4: Relationship between traveling waves of forest and savanna with tanh-like initial condition.

3.2 sech-like Initial Condition

With parameters $a = 0.4, b = 3, \mu = 0.8, \tau = 0.1, r = 0.17, \alpha = 1.0, \beta = 1.3, \delta = 0.01$, and sech-like initial conditions: $c_0 = 0, F_0 = 0.8 * (sech((x - L/2))), S_0 = 1.3 * (1 - 0.5 * sech((x - L/2)))$, we can solve the Forest-Savanna model (Figure 5, Figure 6, and Figure 7):

4 Analyze Stability of Forest-Savanna Equations

To analyze the stability of the given equations, we generate the eigenvalues of the Jacobian matrix. We don't need c value here, thus, we generate a new matrix called "Jacobian-main".

$$Jacobian - main = \begin{pmatrix} D_F E_F & D_S E_F \\ D_F E_S & D_S E_S \end{pmatrix}$$

which $D_F E_F = cD_1 - \delta D_2 + \alpha - 2\mu spdiags(F, 0, Nx, Nx) - aspdiags(S, 0, Nx, Nx)$, $D_S E_F = aspdiags(F, 0, Nx, Nx)$, $D_F E_S = -bspdiags(S, 0, Nx, Nx)$, $D_S E_S = cD_1 - D_2 + \beta - 2spdiags(S, 0, Nx, Nx) - bspdiags(F, 0, Nx, Nx)$.

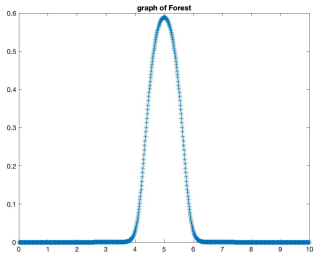

Figure 5: Traveling wave of forest. From the graph, we can tell that the forest grows from F = 0 and finally reaches a peak state at F = 0.8.

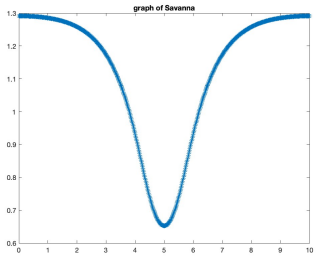

Figure 6: Traveling wave of savanna. From the graph, we can tell that the savanna grows from S = 1.3 and finally reaches a valley between S = 0.6 and S = 0.7.

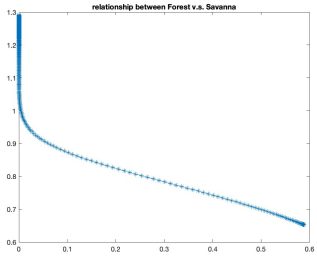

Figure 7: Relationship between traveling waves of forest and savanna with sech-like initial condition.

We compute 100 eigenvalues that closest to 0, expressed them in a 100×1 array:

$$lambda = eigs(-J_M, 100, 0);$$

Plot the Jacobian with x-axis represents the real parts of the eigenvalue, and y-axis represents the imaginary parts of the eigenvalue.
If all eigenvalues have a negative real part, i.e. located on the right side of the graph, then we conclude that equation (2) is stable under the given conditions. If the eigenvalue(s) have a positive real part, i.e. located on the left side of the graph, then we conclude that equation (2) is unstable under the given conditions.

4.1 tanh-like Initial Condition

With parameters $a = 1.3, b = 2, m = 0.02, n = 0.4, \mu = 0.1, \delta = 0.0001$, and tanh-like initial condition: $c_0 = 2, F_0 = 0.8 * (0.5 + 0.5 * tanh((x - L/2)))$, $S_0 = 0.6 * (0.5 - 0.5 * tanh((x - L/2)))$, In order to make the model more research-able, we would like to have unstable solutions.

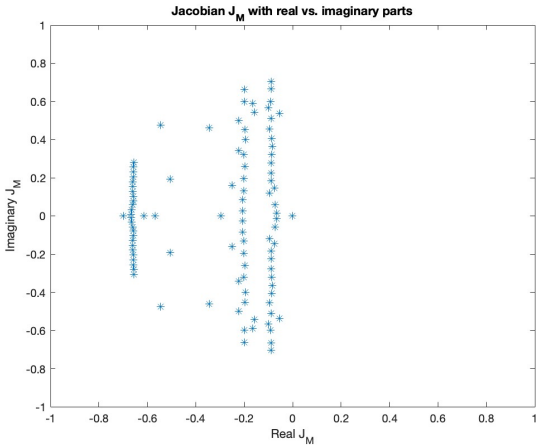

Figure 8: Jacobian plot with real parts of eigenvalues on the x-axis and imaginary parts on the y-axis. Since all eigenvalues with negative real parts, solution of the Forest-Savanna model with given parameters and tanh-like initial conditions is stable.

4.2 sech-like Initial Condition

With parameters $a = 0.4, b = 3, \mu = 0.8, \tau = 0.1, r = 0.17, \alpha = 1.0, \beta = 1.3, \delta = 0.01$, and sech-like initial conditions: $c_0 = 0, F_0 = 0.8 * (sech((x - L/2))), S_0 = 1.3 * (1 - 0.5 * sech((x - L/2)))$,

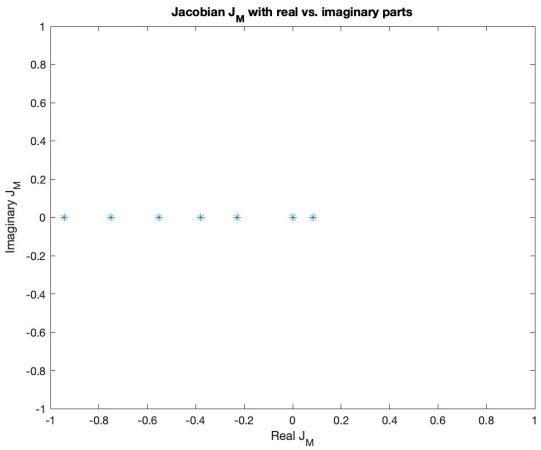

Figure 9: Jacobian plot with real parts of eigenvalues on the x-axis and imaginary parts on the y-axis. Since there is a eigenvalue with positive real pars, solution of the Forest-Savanna model with given parameters and sech-like initial conditions is unstable.

5 Prospective

Since the Forest-Savanna model with sech-like initial condition is unstable, we can further research on its instability and exploring its 3-D pattern with different parameters.

6 MATLAB Code

6.1 Solving Newton's Method by Jacobian

```
format long
options = optimset('Jacobian', 'on');

% 1:
x0 = 1;
fsolve(@(x) f1(x), x0, options)


% 2:
U0 = [1;1];
fsolve(@(U) f2(U), U0, options)

function [f1, J1] = f1(x)
    f1 = x^2-3;
    J1 = 2*x;
end
%%
function [f2, J2] = f2(U)
    x = U(1);
    y = U(2);
    f2 = [x^2-3; x^2+y^2-4];
    J2 = [2*x, 0; 2*x, 2*y];
end
```

6.2 Traveling Waves

```
    format long
% Define time-space
L = 50; N = 500;
tspan = 0:0.1:10
deltax = L/(N-1)

% Define parameters
a = 1.3; b = 2; m = 0.02; n = 0.4; mu = 0.1;
alpha = mu - m;
beta = 1 - n;
delta = 0.0001;
c0 = 2;

% Discretize space
x = (transpose(0:(N-1))) * deltax

F0 = @(x) 0.8*(0.5+0.5*tanh((x-L/2))); % self-defined!!!!
S0 = @(x) 0.6*(0.5-0.5*tanh((x-L/2))); % self-defined!!!!

% Define Initial Condition
U0 = [F0(x); S0(x); c0]; % size = 2N+1

% Define D2!!!!!
D2 = zeros(N,N);
D2(1,1) = -2; D2(1,2) = 2;
```

```
    for i = 2:N-1
        D2(i,i) = -2;
        D2(i,i-1) = 1;
        D2(i,i+1) = 1;
    end
D2(N,N) = -2; D2(N,N-1) = 2;
D2 = D2/deltax^2;
D2 = sparse(D2);

% Define D1
D1 = zeros(N,N);
D1(1,2) = 0;
for i = 2:N-1
        D1(i,i) = 0;
        D1(i,i-1) = -1;
        D1(i,i+1) = 1;
end
D1(N,N-1) = 0;
D1 = D1/(2*deltax);
D1 = sparse(D1);

options = optimset('Jacobian', 'on', 'display', 'iter');
% output the solution
FS = fsolve(@(U) FS_function(U, N, U0, D1, D2, alpha, beta, delta, mu, a, b), U0, options)
Obtain the solution
% define solution of F and S, and the speed c seperately
F_solution = FS(1:N);
S_solution = FS(N+1:2*N);
c_solution = FS(2*N+1);

% Plot F (Forest)
F_solution = FS(1:N);
plot(x, F_solution, '-*')
title('graph of Forest')
% Plot S (Savanna)
S_solution = FS(N+1:2*N);
plot(x, S_solution, '-*')
title('graph of Savanna')
% F vs S
plot(FS(1:N), FS(N+1:2*N), '-*')
title('relationship between Forest v.s. Savanna')

%%

function [FS_function, J] = FS_function(U, N, U0, D1, D2, alpha, beta, delta, mu, a, b)
    F = U(1:N);
    S = U((N+1):(2*N));
    c = U(2*N+1);
    F0 = U0(1:N);
    S0 = U0((N+1):(2*N));
    % Construct functions
    FS_function = [c*D1*F - delta*D2*F - alpha*F + mu*F.*F + a*S.*F;
                   c*D1*S - D2*S - beta*S + S.*S + b*F.*S;
                   transpose([D1*F;D1*S])*([F;S]-[F0;S0])];
```

```
    % Construct Jacobian
    J = zeros(2*N+1,2*N+1);
        J(1:N, 1:N) = c*D1 - delta*D2 +spdiags(2*mu*F-alpha, 0, N, N) + a*spdiags(S, 0, N, N);
        J(1:N, N+1:2*N) = a*spdiags(F, 0, N, N);
        J(1:N, 2*N+1) = D1*F;

        J(N+1:2*N, 1:N) = b*spdiags(S, 0, N, N);
        J(N+1:2*N, N+1:2*N) = c*D1 - D2 + spdiags(2*S-beta, 0, N, N) + b*spdiags(F, 0, N, N);
        J(N+1:2*N, 2*N+1) = D1*S;

        J(2*N+1, 1:N) = transpose(D1*F) + transpose(F-F0)*D1;
        J(2*N+1, N+1:2*N) = transpose(D1*S) + transpose(S-S0)*D1;
end
```

6.3 Stability Analysis

```
% define the "main" Jacobian J_M as a 2Nx2N matrix
J_M = zeros(2*N, 2*N);
    % row 1st half: forest
    J_M(1:N, 1:N) = c_solution*D1 - delta*D2 +spdiags(2*mu*F_solution-alpha, 0, N, N) + a*spdiags(S_solu
    J_M(1:N, N+1:2*N) = a*spdiags(F_solution, 0, N, N);
    % row 2nd half: savanna
    J_M(N+1:2*N, 1:N) = b*spdiags(S_solution, 0, N, N);
    J_M(N+1:2*N, N+1:2*N) = c_solution*D1 - D2 + spdiags(2*S_solution-beta, 0, N, N) + b*spdiags(F_solu
J_M
% compute 100 eigenvalues closet to 0
lambda = eigs(-J_M, 100, 0);
% seperate real and imaginary parts of eigenvalues
lambda_real = real(lambda)
lambda_imag = imag(lambda)

tic
for i = 1:1:100
    plot(lambda_real(:,:), lambda_imag(:,:), '*');
    title('Jacobian J_M with real vs. imaginary parts');
    xlabel('Real J_M')
    ylabel('Imaginary J_M')
    axis([-1 1 -1 1]);
end
toc
```

7 Reference

Banerjee, Swarnendu et al. *Rethinking tipping points in spatial ecosystems.* Cornell University. 23 June 2023. https://arxiv.org/abs/2306.13571.

Jonkhout, C.J.H. *Traveling wave solutions of reaction-diffusion equations in population dynamics.* July 27, 2016. https://www.universiteitleiden.nl/binaries/content/assets/science/mi/scripties/jonkhout.pdf

Banerjee, Swarnendu, et al. *Rethinking tipping points in spatial ecosystems.* June 23, 2023. https://arxiv.org/abs/2306.13571.