SUPPLEMENTARY INFORMATION

Predicting outcomes of competition between two planktonic primary producers along vertical opposing resource gradients using eigenvalue estimates

Arthur F. Rossignol[1,2] and Dr. Sabine Wollrab[3*]

[1] Leibniz Institute of Freshwater Ecology and Inland Fisheries, Department of Plankton and Microbial Ecology, Zur alten Fischerhütte 2, 16775 Stechlin, Germany

[2] École polytechnique, IP Paris, Route de Saclay, 91120 Palaiseau, France

* Corresponding author: *sabine.wollrab@igb-berlin.de*

published in *[...]*

Contents

1. Algorithm of the prediction procedure 2
2. Biomass profiles related to the two-parameter bifurcation plot 3
3. Effects of parameters on eigenvalue estimation and prediction performance 5
4. MATLAB codes 8

1. Algorithm of the prediction method

```
compute λ1*
compute λ2*
if λ1* ≤ 0 then
    if λ2* ≤ 0 then
        return 𝓔0
    else
        return 𝓔2
else
    if λ2* ≤ 0 then
        return 𝓔1
    else
        compute the S1's monoculture non-trivial steady state (Â1, N̂1)
        compute Λ1*
        compute the S2's monoculture non-trivial steady state (Â2, N̂2)
        compute Λ2*
        if Λ1* ≤ 0 then
            if Λ2* ≤ 0 then
                return "bistability" (𝓔1 or 𝓔2)
            else
                return 𝓔2
        else
            if Λ2* ≤ 0 then
                return 𝓔3
            else
                return 𝓔1
```

2. Biomass profiles related to the two-parameter bifurcation plots

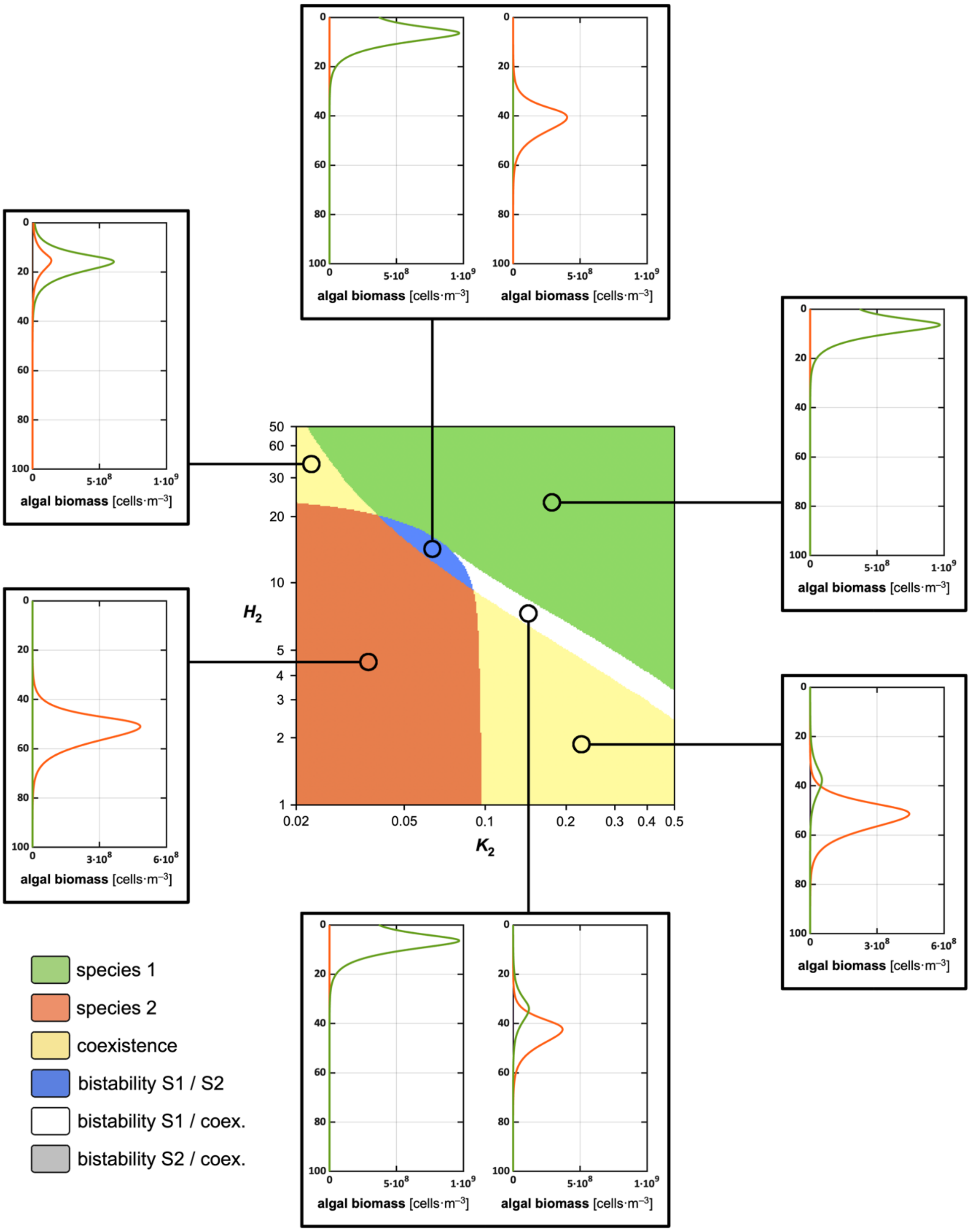

Figure S1. Examples of equilibrium biomass profiles related to Fig. 3.1b with respect to two-parameter space (H_2, K_2) with H_2 the half-saturation constant of light uptake and K_2 the half-saturation constant of nutrient uptake of S2. Outcome results are obtained from the competition simulation. In biomass profile subplots, the green line represents S1's biomass and the red line represents S2's biomass. In bistability cases, the left subplot is obtained with S1 as the resident and S2 as the invader, while the right subplot is obtained with S1 as the invader and S2 as the resident. Parameters from Tab. 1 with $D = 0.1$ and $r = 0.5$.

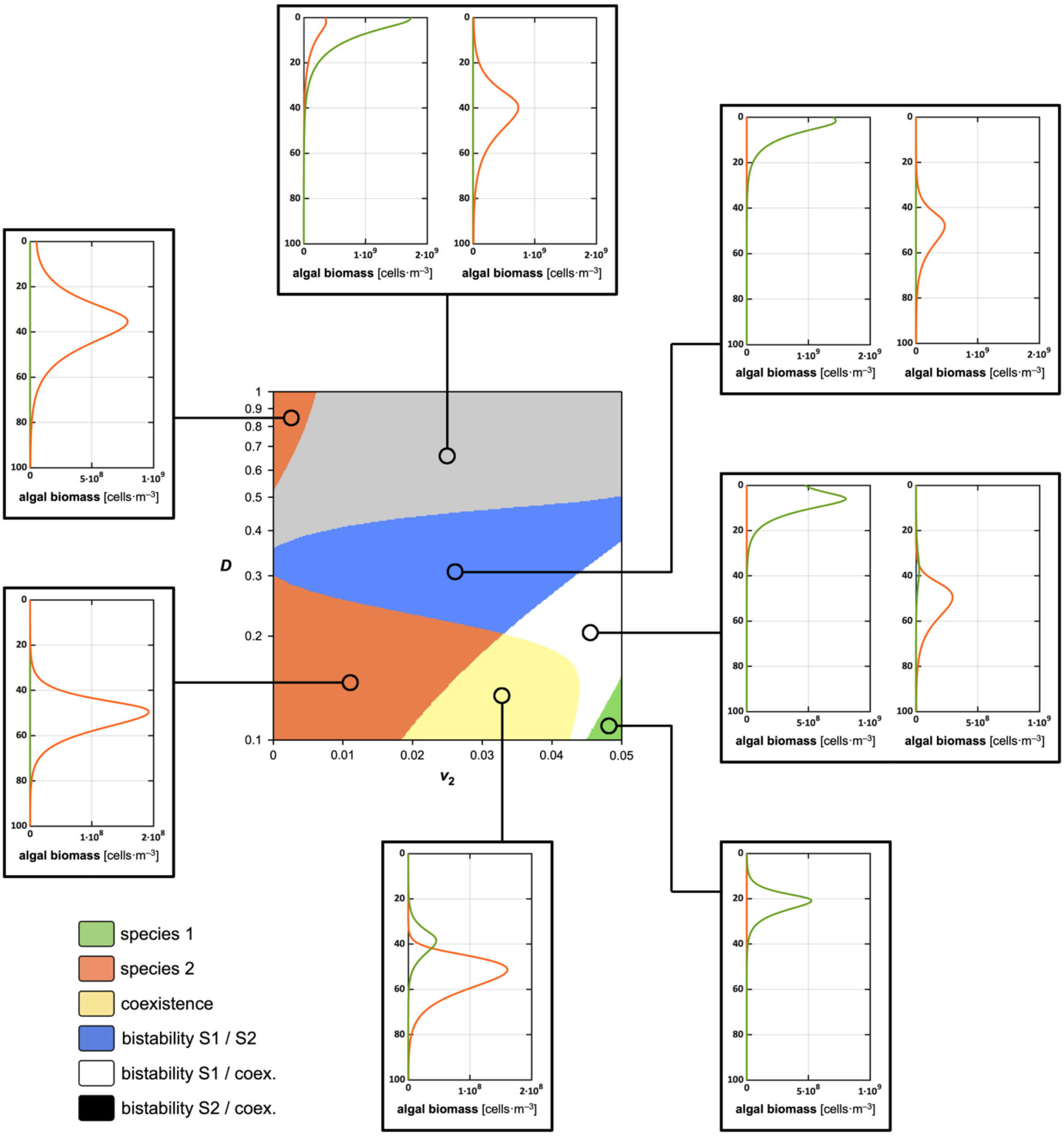

Figure S2. Examples of equilibrium biomass profiles related to Fig. 3.2b with respect to two-parameter space (υ_2, D) with υ_2 the sinking velocity of S2 and D the eddy diffusion coefficient. Outcome results are obtained from the competition simulation. In biomass profile subplots, the green line represents S1's biomass and the red line represents S2's biomass. In bistability cases, the left subplot is obtained with S1 as the resident and S2 as the invader, while the right subplot is obtained with S1 as the invader and S2 as the resident. Parameters from Tab. 1 with $r = 0$.

3 Effects of parameters on eigenvalue estimation and prediction performance

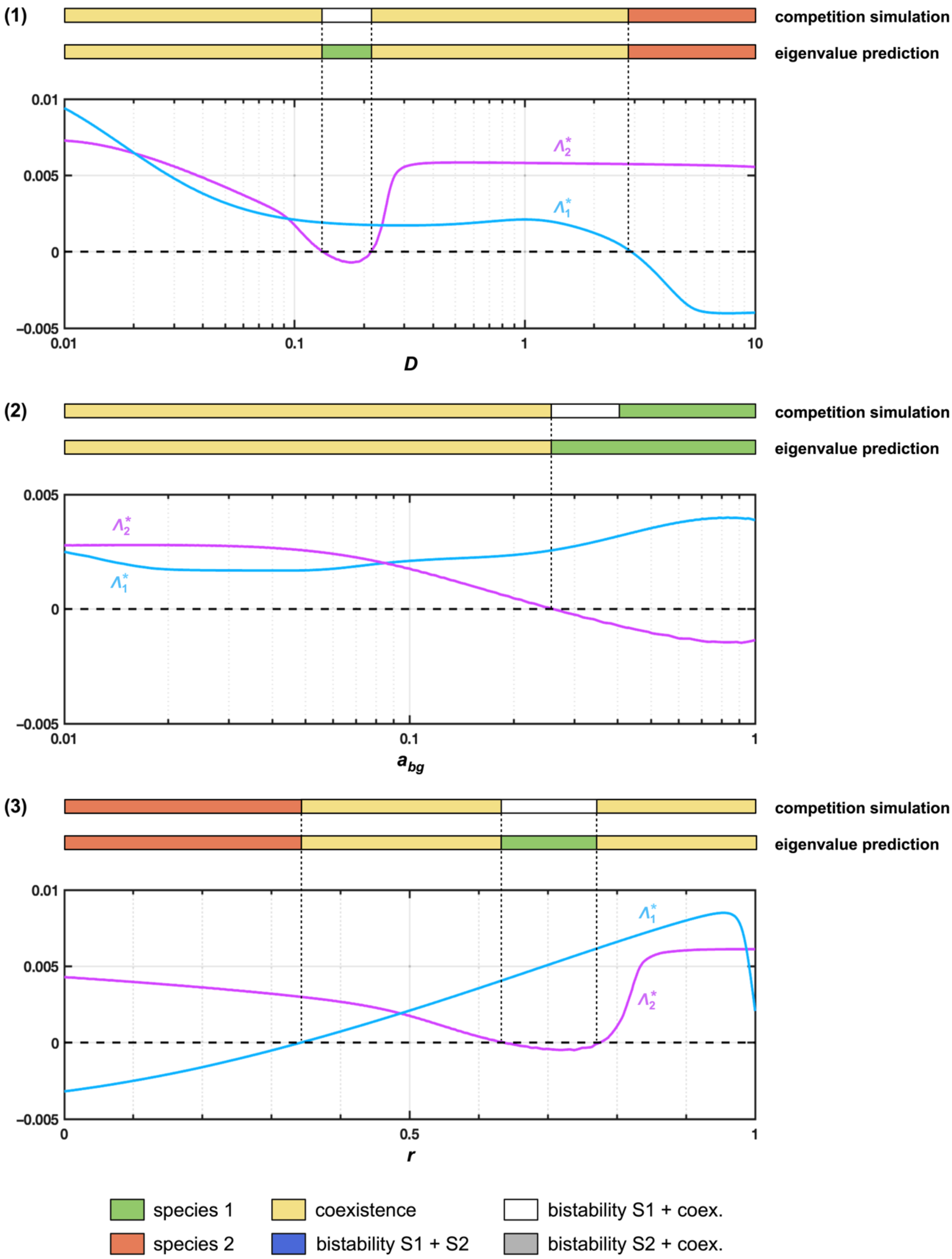

Figure S3. Effect of gradients of eddy diffusion coefficient D (panel **1**), background turbidity a_{bg} (panel **2**), recycling rate r, (panel **3**) on principal eigenvalues Λ_1^* and Λ_2^*. In each panel, the subplot shows the eigenvalues Λ_1^* and Λ_2^* as functions of the parameter of interest, and the horizontal color bars respectively indicate the competition outcome results from either the competition simulation or the eigenvalue prediction. Parameters from Tab. 1 with $D = 0.1$ and $r = 0.5$.

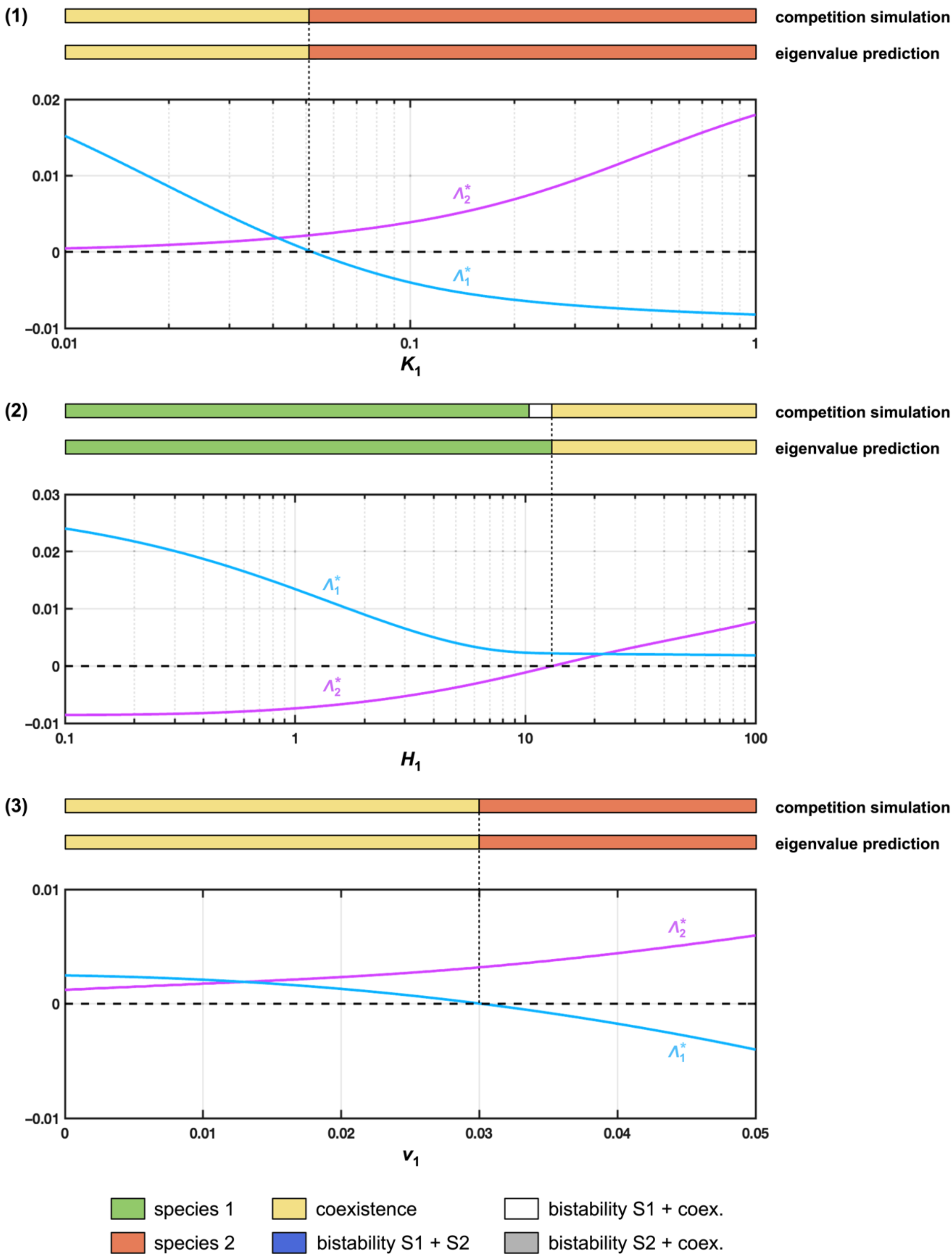

Figure S4. Effect of gradients of half-saturation constant K_1 (panel **1**), half-saturation constant H_1 (panel **2**), sinking velocity v_1 (panel **3**) on principal eigenvalues Λ_1^* and Λ_2^*. In each panel, the subplot shows the eigenvalues Λ_1^* and Λ_2^* as functions of the parameter of interest, and the horizontal color bars respectively indicate the competition outcome results from either the competition simulation or the eigenvalue prediction. Parameters from Tab. 1 with $D = 0.1$ and $r = 0.5$.

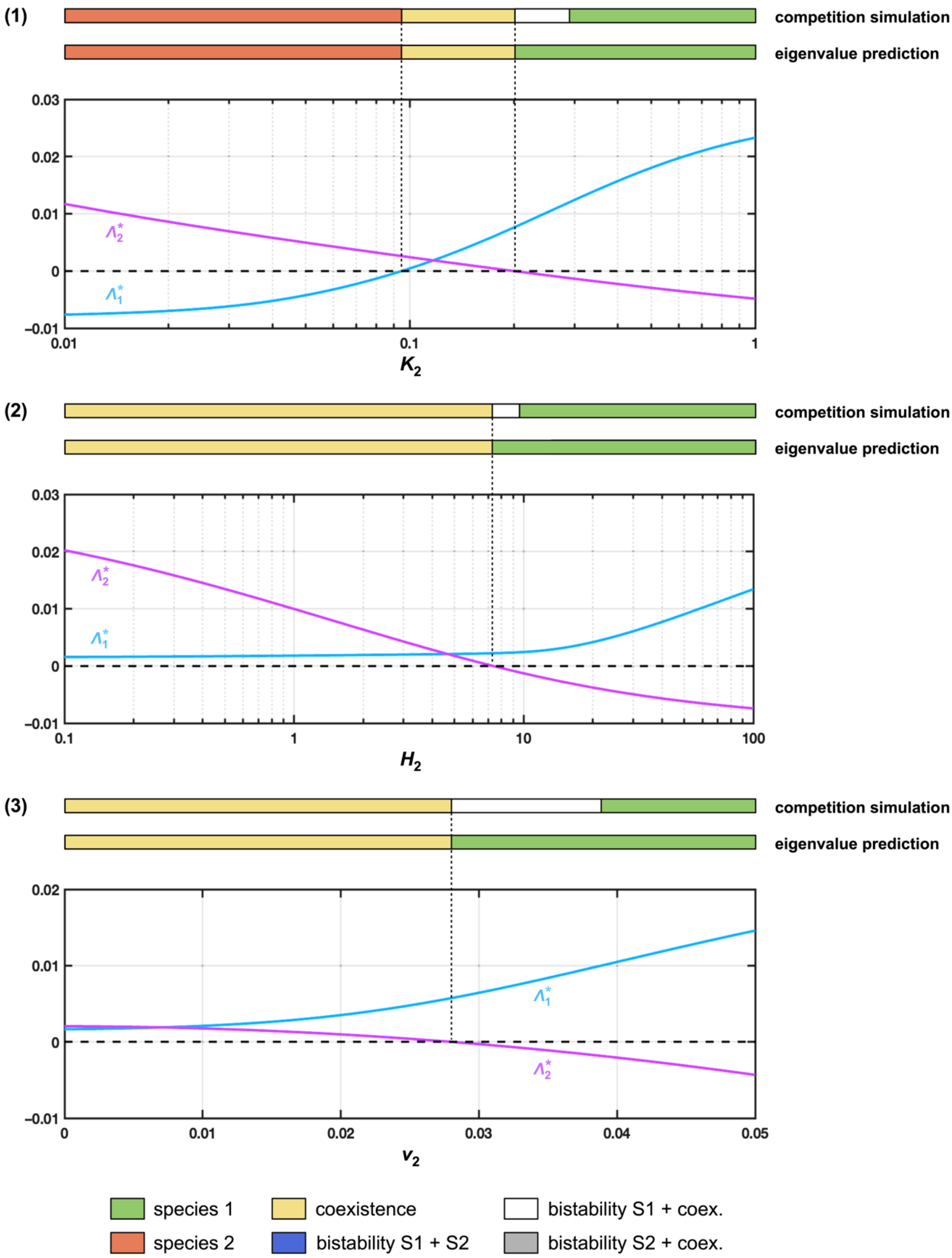

Figure S5. Effect of gradients of half-saturation constant K_2 (panel **1**), half-saturation constant H_2 (panel **2**), sinking velocity v_2 (panel **3**) on principal eigenvalues Λ_1^* and Λ_2^*. In each panel, the subplot shows the eigenvalues Λ_1^* and Λ_2^* as functions of the parameter of interest, and the horizontal color bars respectively indicate the competition outcome results from either the competition simulation or the eigenvalue prediction. Parameters from Tab. 1 with $D = 0.1$ and $r = 0.5$.

4. MATLAB codes

The MATLAB codes are:

- `run.m`
- `one_species.m`
- `two_species.m`
- `growth_rate.m`
- `principal_eigenvalue.m`

They are also readily available in the following GitHub repository:
https://github.com/arthur-f-rossignol/article-002

run.m

```
%% Main script

% abiotic parameters
l    = 100;          % maximum depth of the water column
D    = 0.1;          % eddy diffusion coefficient
a_bg = 0.1;          % background turbidity
r    = 0.5;          % recycling rate
E    = 0.1;          % sediment interface permeability
N_0  = 10;           % nutrient concentration in the sediments
I_0  = 600;          % light intensity at the surface

% algal parameters of S1
mu_1 = 0.04;         % maximum growth rate
K_1  = 0.04;         % half-saturation constant for nutrient uptake
H_1  = 20;           % half-saturation constant for light uptake
m_1  = 0.01;         % mortality rate
v_1  = 0.01;         % sinking velocity
q_1  = 2 * 1e-10;    % internal nutrient quota
a_1  = 6 * 1e-10;    % absorption coefficient of biomass

% algal parameters of S2
mu_2 = 0.04;         % maximum growth rate
K_2  = 0.12;         % half-saturation constant for nutrient uptake
H_2  = 5;            % half-saturation constant for light uptake
m_2  = 0.01;         % mortality rate
v_2  = 0.01;         % sinking velocity
q_2  = 5 * 1e-10;    % internal nutrient quota
a_2  = 1 * 1e-10;    % absorption coefficient of biomass

% spatial discretization
dz = 0.25;
n  = floor(l / dz);
Z  = linspace(0, l, n);

% time
t_max = 1e8;

% solver options
option = odeset('nonnegative', 1, 'RelTol', 1e-8, 'AbsTol', 1e-10);

% resource profiles of trivial steady states
N = N_0 * ones(1, n);
I = zeros(n, 1);
I(1) = I_0 * exp(- a_bg * 0.5 * dz);
for i = 2:n
    I(i) = I_0 * exp(- a_bg * (i - 0.5) * dz);
end

% eigenvalues λ₁* and λ₂*
lambda_1 = principal_eigenvalue(N, I, [n, dz, D, mu_1, K_1, H_1, m_1, v_1]);
lambda_2 = principal_eigenvalue(N, I, [n, dz, D, mu_2, K_2, H_2, m_2, v_2]);

% integration of S1's monoculture
parameters = [n, dz, D, N_0, I_0, a_bg, E, r, mu_1, K_1, H_1, m_1, v_1, q_1, a_1];
U0 = [1e3 * ones(n, 1);
      transpose(linspace(0, N_0, n))];
[t, monoculture_1] = ode15s(@(t, U) one_species(t, U, parameters), ...
                            [0, t_max], ...
                            U0, ...
                            option);

% biomass and resource profiles from S1's monoculture
A1 = monoculture_1(end, 1:n);
N1 = monoculture_1(end, (n + 1):(2 * n));
```

```matlab
I1 = zeros(n, 1);
I1(1) = I_0 * exp(- a_bg * 0.5 * dz ...
                  - a_1 * ((3 * A1(1) - A1(2)) / 8 + 3 * A1(1) / 4) * dz);
for i = 2:n
    S = - a_1 * ((3 * A1(1) - A1(2)) / 8 + 3 * A1(1) / 4) * dz;
    for k = 2:(i - 1)
        S = S - a_1 * A1(k) * dz;
    end
    I1(i) = I_0 * exp(S - a_bg * (i - 0.5) * dz ...
                        - a_1 * A1(i) * 0.5 * dz);
end

% eigenvalues Λ₂*
LAMBDA_2 = principal_eigenvalue(N1, I1, [n, dz, D, mu_2, K_2, H_2, m_2, v_2]);

% integration of S2's monoculture
parameters = [n, dz, D, N_0, I_0, a_bg, E, r, mu_2, K_2, H_2, m_2, v_2, q_2, a_2];
U0 = [1e3 * ones(n, 1);
      transpose(linspace(0, N_0, n))];
[t, monoculture_2] = ode15s(@(t, U) one_species(t, U, parameters), ...
                            [0, t_max], ...
                            U0, ...
                            option);

% biomass and resource profiles from S2's monoculture
A2 = monoculture_2(end, 1:n);
N2 = monoculture_2(end, (n + 1):(2 * n));
I2 = zeros(n, 1);
I2(1) = I_0 * exp(- a_bg * 0.5 * dz ...
                  - a_2 * ((3 * A2(1) - A2(2)) / 8 + 3 * A2(1) / 4) * dz);
for i = 2:n
    S = - a_2 * ((3 * A2(1) - A2(2)) / 8 + 3 * A2(1) / 4) * dz;
    for k = 2:(i - 1)
        S = S - a_2 * A2(k) * dz;
    end
    I2(i) = I_0 * exp(S - a_bg * (i - 0.5) * dz ...
                        - a_2 * A2(i) * 0.5 * dz);
end

% eigenvalues Λ₁*
LAMBDA_1 = principal_eigenvalue(N2, I2, [n, dz, D, mu_1, K_1, H_1, m_1, v_1]);

% competition simulation with S1 as resident
parameters = [n, dz, D, N_0, I_0, a_bg, E, r, ...
              mu_1, K_1, H_1, m_1, v_1, q_1, a_1...
              mu_2, K_2, H_2, m_2, v_2, q_2, a_2];
U0 = [1e3 * ones(n, 1);
      1e-3 * ones(n, 1);
      transpose(linspace(0, N_0, n))];
[t, competition_1] = ode15s(@(t, U) two_species(t, U, parameters), ...
                            [0, t_max], ...
                            U0, ...
                            option);

% biomass and resource profiles from competition with S1 as resident
A1_comp_1 = competition_1(end, 1:n);
A2_comp_1 = competition_1(end, (n + 1):(2 * n));
N_comp_1  = competition_1(end, (2 * n + 1):(3 * n));

% competition simulation with S2 as resident
U0 = [1e-3 * ones(n, 1);
      1e3 * ones(n, 1);
      transpose(linspace(0, N_0, n))];
[t, competition_2] = ode15s(@(t, U) two_species(t, U, parameters), ...
                            [0, t_max], ...
                            U0, ...
```

```
                            option);

% biomass and resource profiles from competition with S2 as resident
A1_comp_2 = competition_2(end, 1:n);
A2_comp_2 = competition_2(end, (n + 1):(2 * n));
N_comp_2  = competition_2(end, (2 * n + 1):(3 * n));
```

one_species.m

```
%% Numerical scheme for the one-species model integration

function dU_dt = one_species(t, U, parameters)

    % definition of parameters
    n    = parameters(1);
    dz   = parameters(2);
    D    = parameters(3);
    N_0  = parameters(4);
    I_0  = parameters(5);
    a_bg = parameters(6);
    E    = parameters(7);
    r    = parameters(8);
    mu   = parameters(9);
    K    = parameters(10);
    H    = parameters(11);
    m    = parameters(12);
    v    = parameters(13);
    q    = parameters(14);
    a    = parameters(15);

    % preallocation of vectors
    dA_dz   = zeros(n, 1);
    dA_dz2  = zeros(n, 1);
    dA_dt   = zeros(n, 1);
    G       = zeros(n, 1);
    dN_dz2  = zeros(n, 1);
    dN_dt   = zeros(n, 1);
    I       = zeros(n, 1);

    % starting values of A and N
    A = U(1:n);
    N = U((n + 1):(2 * n));

    % light intensity
    I(1) = I_0 * exp(- a_bg * 0.5 * dz ...
                 - a * ((3 * A(1) - A(2)) / 8 + 3 * A(1) / 4) * dz);
    for i = 2:n
        S = - a * ((3 * A(1) - A(2)) / 8 + 3 * A(1) / 4) * dz;
        for k = 2:(i - 1)
            S = S - a * A(k) * dz;
        end
        I(i) = I_0 * exp(S - a_bg * (i - 0.5) * dz ...
                       - a * A(i) * 0.5 * dz);
    end

    % 1st spatial derivative of A (upwind 3rd-order scheme)
    dA_dz(3:(n - 1)) = (2 * A(4:n) ...
                     + 3 * A(3:(n - 1)) ...
                     - 6 * A(2:(n - 2)) ...
                     + A(1:(n - 3))) / (6 * dz);
    dA_dz(1)         = (A(1) + A(2)) / (2 * dz);
    dA_dz(2)         = (- A(1) + 5 * A(2) + 2 * A(3)) / (6 * dz) ...
                     - (A(1) + A(2)) / (2 * dz);
    dA_dz(n)         = (A(n - 2) - 5 * A(n - 1) - 2 * A(n)) / (6 * dz);

    % 2nd spatial derivative of A (symmetrical 2nd-order scheme)
    dA_dz2(2:(n - 1)) = (A(3:n) ...
                      - A(2:(n - 1)) ...
                      - A(2:(n - 1)) ...
                      + A(1:(n - 2))) / dz^2;
    dA_dz2(1)         = (A(2) - A(1)) / dz^2;
    dA_dz2(n)         = (A(n - 1) - A(n)) / dz^2;
```

```
    % 2nd spatial derivative of B (symmetrical 2nd-order scheme)
    dN_dz2(2:(n - 1)) = (N(3:n) ...
                      - N(2:(n - 1)) ...
                      - N(2:(n - 1)) ...
                      + N(1:(n - 2))) / dz^2;
    dN_dz2(1)         = (N(2) - N(1)) / dz^2;
    dN_dz2(n)         = E * (N_0 - N(n)) / dz - (N(n) - N(n - 1)) / dz^2;

    % growth rate
    for i = 1:n
        G(i) = growth_rate(N(i), I(i), mu, K, H);
    end

    % time derivatives of A and N
    dA_dt(1:n) = (G(1:n) - m) .* A(1:n) ...
               - v * dA_dz(1:n) ...
               + D * dA_dz2(1:n);
    dN_dt(1:n) = q * (r * m - G(1:n)) .* A(1:n) ...
               + D * dN_dz2(1:n);

    % output
    dU_dt = [dA_dt; dN_dt];

end
```

two_species.m

```
%% Numerical scheme for the two-species model integration

function dU_dt = two_species(t, U, parameters)

    % definition of parameters
    n      = parameters(1);
    dz     = parameters(2);
    D      = parameters(3);
    N_0    = parameters(4);
    I_0    = parameters(5);
    a_bg   = parameters(6);
    E      = parameters(7);
    r      = parameters(8);
    mu_1   = parameters(9);
    K_1    = parameters(10);
    H_1    = parameters(11);
    m_1    = parameters(12);
    v_1    = parameters(13);
    q_1    = parameters(14);
    a_1    = parameters(15);
    mu_2   = parameters(16);
    K_2    = parameters(17);
    H_2    = parameters(18);
    m_2    = parameters(19);
    v_2    = parameters(20);
    q_2    = parameters(21);
    a_2    = parameters(22);

    % preallocation of vectors
    dA1_dz  = zeros(n, 1);
    dA1_dz2 = zeros(n, 1);
    dA1_dt  = zeros(n, 1);
    G1      = zeros(n, 1);
    dA2_dz  = zeros(n, 1);
    dA2_dz2 = zeros(n, 1);
    dA2_dt  = zeros(n, 1);
    G2      = zeros(n, 1);
    dN_dz2  = zeros(n, 1);
    dN_dt   = zeros(n, 1);
    I       = zeros(n, 1);

    % starting values of A1, A2, N
    A1 = U(1:n);
    A2 = U((n + 1):(2 * n));
    N  = U((2 * n + 1):(3 * n));

    % computation of light intensity
    I(1) = I_0 * exp(- a_bg * 0.5 * dz ...
                 - a_1 * ((3 * A1(1) - A1(2)) / 8 + 3 * A1(1) / 4) * dz ...
                 - a_2 * ((3 * A2(1) - A2(2)) / 8 + 3 * A2(1) / 4) * dz);
    for i = 2:n
        S = - a_1 * ((3 * A1(1) - A1(2)) / 8 + 3 * A1(1) / 4) * dz ...
            - a_2 * ((3 * A2(1) - A2(2)) / 8 + 3 * A2(1) / 4) * dz;
        for k = 2:(i - 1)
            S = S - a_1 * A1(k) * dz ...
                  - a_2 * A2(k) * dz;
        end
        I(i) = I_0 * exp(S - a_bg * (i - 0.5) * dz ...
                           - a_1 * A1(i) * 0.5 * dz ...
                           - a_2 * A2(i) * 0.5 * dz);
    end

    % 1st spatial derivative of A1 (upwind 3rd-order scheme)
    dA1_dz(3:(n - 1)) = (2 * A1(4:n) ...
```

```
                      + 3 * A1(3:(n - 1)) ...
                      - 6 * A1(2:(n - 2)) ...
                      + A1(1:(n - 3))) / (6 * dz);
    dA1_dz(1)         = (A1(1) + A1(2)) / (2 * dz);
    dA1_dz(2)         = (- A1(1) + 5 * A1(2) + 2 * A1(3)) / (6 * dz) ...
                      - (A1(1) + A1(2)) / (2 * dz);
    dA1_dz(n)         = (A1(n - 2) - 5 * A1(n - 1) - 2 * A1(n)) / (6 * dz);

    % 1st spatial derivative of A2 (upwind 3rd-order scheme)
    dA2_dz(3:(n - 1)) = (2 * A2(4:n) ...
                      + 3 * A2(3:(n - 1)) ...
                      - 6 * A2(2:(n - 2)) ...
                      + A2(1:(n - 3))) / (6 * dz);
    dA2_dz(1)         = (A2(1) + A2(2)) / (2 * dz);
    dA2_dz(2)         = (- A2(1) + 5 * A2(2) + 2 * A2(3)) / (6 * dz)  ...
                      - (A2(1) + A2(2)) / (2 * dz);
    dA2_dz(n)         = (A2(n - 2) - 5 * A2(n - 1) - 2 * A2(n)) / (6 * dz);

    % 2nd spatial derivative of A1 (symmetrical 2nd-order scheme)
    dA1_dz2(2:(n - 1)) = (A1(3:n) ...
                       - A1(2:(n - 1)) ...
                       - A1(2:(n - 1)) ...
                       + A1(1:(n - 2))) / dz^2;
    dA1_dz2(1)         = (A1(2) - A1(1)) / dz^2;
    dA1_dz2(n)         = (A1(n - 1) - A1(n)) / dz^2;

    % 2nd spatial derivative of A2 (symmetrical 2nd-order scheme)
    dA2_dz2(2:(n - 1)) = (A2(3:n) ...
                       - A2(2:(n - 1)) ...
                       - A2(2:(n - 1)) ...
                       + A2(1:(n - 2))) / dz^2;
    dA2_dz2(1)         = (A2(2) - A2(1)) / dz^2;
    dA2_dz2(n)         = (A2(n - 1) - A2(n)) / dz^2;

    % 2nd spatial derivative of N (symmetrical 2nd-order scheme)
    dN_dz2(2:(n - 1)) = (N(3:n) ...
                      - N(2:(n - 1)) ...
                      - N(2:(n - 1)) ...
                      + N(1:(n - 2))) / dz^2;
    dN_dz2(1)         = (N(2) - N(1)) / dz^2;
    dN_dz2(n)         = E * (N_0 - N(n)) / dz - (N(n) - N(n - 1)) / dz^2;

    % growth rate
    for i = 1:n
        G1(i) = growth_rate(N(i), I(i), mu_1, K_1, H_1);
        G2(i) = growth_rate(N(i), I(i), mu_2, K_2, H_2);
    end

    % time derivatives of A and N
    dA1_dt(1:n) = (G1(1:n) - m_1) .* A1(1:n) ...
                - v_1 * dA1_dz(1:n) ...
                + D * dA1_dz2(1:n);
    dA2_dt(1:n) = (G2(1:n) - m_2) .* A2(1:n) ...
                - v_2 * dA2_dz(1:n) ...
                + D * dA2_dz2(1:n);
    dN_dt(1:n)  = q_1 * (r * m_1 - G1(1:n)) .* A1(1:n) ...
                + q_2 * (r * m_2 - G2(1:n)) .* A2(1:n) ...
                + D * dN_dz2(1:n);

    % output
    dU_dt = [dA1_dt; dA2_dt; dN_dt];

end
```

growth_rate.m

```
%% Function computing the growth rate

function g = growth_rate(N, I, mu, K, H)

    % Monod function for nutrient use efficiency
    N_monod = N / (K + N);

    % Monod function for light use efficiency
    I_monod = I / (H + I);

    % Liebig's law of the minimum
    g = mu * min(N_monod, I_monod);

end
```

principal_eigenvalue.m

```
%% Function computing the principal eigenvalue estimate

function p_eig = principal_eigenvalue(N, I, parameters)

    % definition of parameters
    n  = parameters(1);
    dz = parameters(2);
    D  = parameters(3);
    mu = parameters(4);
    K  = parameters(5);
    H  = parameters(6);
    m  = parameters(7);
    v  = parameters(8);

    % preallocation of matrices
    P = zeros(n, n);
    Q = zeros(n, n);

    % boundary conditions
    alpha = 2 - (8 * D) / (4 * D + v * dz);
    beta  = 2 - (8 * D) / (4 * D - v * dz);

    % matrix P
    for i = 1:n
        P(i, i) = 2 * D;
    end
    P(1, 1) = (alpha + 1) * D;
    P(n, n) = (beta + 1) * D;
    for i = 1:(n - 1)
        P(i + 1, i) = - D;
        P(i, i + 1) = - D;
    end
    P = P / dz^2;

    % matrix Q
    for i = 1:n
        Q(i, i) = m + (v^2 / (4 * D)) - growth_rate(N(i), I(i), mu, K, H);
    end

    % spectrum and principal eigenvalue
    M = P + Q;
    spectrum = - eig(M);
    p_eig = max(spectrum);

end
```