

Killer Yeast Vs. Sensitive Yeast

Evan Cummings Intizor Aliyorov

Malachi J. Cryder

MATH 445 - Statistical, Dynamical, and Computational Modeling

December 9, 2013

Proposal

The differential equation we may use for modeling the growth of yeast is the same as that used for bacterial growth in a chemostat:

$$\begin{aligned}\frac{dN}{dt} &= k(C)N - \frac{FN}{V}, \\ \frac{dC}{dt} &= -\alpha k(C)N - \frac{FC}{V} + \frac{FC_0}{V},\end{aligned}$$

with initial conditions $C(0) = C_i$ and $N(0) = N_i$, N is the unitless optical density of yeast in the chamber, C is the unitless optical density of nutrient in the chamber, C_0 is the unitless optical density of nutrient in the reservoir, F is the in/out volume flow rate with units volume/time, V is the volume of the chamber, α is a unitless inverse of the yield constant, and $k(C)$ is the reproduction rate for yeast in units 1/time with possible formula chosen such that $\lim_{C \rightarrow \infty} k(C) = k_{max}$, and k_{max} represents the maximum possible reproduction rate:

$$k(C) = \frac{k_{max}C}{C_n + C}.$$

where C_n is chosen such that $k(C_n) = k_{max}/2$. Because the concentration in the tank $C(t)$ is related to the concentration in the reservoir by $C(t) \leq C_0$, C_0 may be chosen sufficiently small such that

$$k(C) = \frac{k_{max}C}{C_n + C} \approx \frac{k_{max}C}{C_n} = KC,$$

where K has units 1/time. The equations we need to solve then become

$$\frac{dN}{dt} = KCN - \frac{FN}{V}, \tag{1}$$

$$\frac{dC}{dt} = -\alpha KCN - \frac{FC}{V} + \frac{FC_0}{V}. \tag{2}$$

The quantitative measurement for fitness is a unitless measurement of optical density at steady state (N) at a given flow rate (F) in volumes/hr. In order to find the steady states, we have to

find the intersections of the null-clines at equilibrium points (\bar{N}, \bar{C}) , i.e. $dN(\bar{N}, \bar{C})/dt = 0$ and $dC(\bar{N}, \bar{C})/dt = 0$:

$$\begin{aligned}\frac{dN(\bar{N}, \bar{C})}{dt} &= K\bar{C}\bar{N} - \frac{F\bar{N}}{V}, \\ &= \bar{N} \left(K\bar{C} - \frac{F}{V} \right) = 0,\end{aligned}$$

which is zero for $\bar{N} = 0$ or $K\bar{C} = F/V$. Solving the other equation gives us the other steady-states:

$$\frac{dC(\bar{N}, \bar{C})}{dt} = -\alpha K\bar{C}\bar{N} - \frac{F\bar{C}}{V} + \frac{FC_0}{V} = 0,$$

which is zero for $\alpha K\bar{C}\bar{N} + \frac{F\bar{C}}{V} = \frac{FC_0}{V}$.

In order to evaluate these null-clines, we need to evaluate the non-trivial cases, here for $\bar{N} = 0$,

$$\begin{aligned}K\bar{C} &= \frac{F}{V} \\ \Rightarrow \bar{C} &= \frac{F}{VK}.\end{aligned}\tag{3}$$

Likewise, for $\dot{C} = 0$,

$$\begin{aligned}\alpha K\bar{C}\bar{N} + \frac{F\bar{C}}{V} &= \frac{FC_0}{V} \\ \Rightarrow \bar{N} &= \frac{FC_0}{V\alpha K\bar{C}} - \frac{F}{V\alpha K} = \frac{F}{V\alpha K} \left(\frac{C_0}{\bar{C}} - 1 \right).\end{aligned}\tag{4}$$

This intersects the $\bar{N} = 0$ nullcline at $\frac{F}{V\alpha K} = 0$ or $\frac{C_0}{\bar{C}} = 1$. However, because F is never 0, we can disregard the first equation, and we know that the only trivial steady-state is located at $\bar{N} = 0$, $\bar{C} = C_0$.

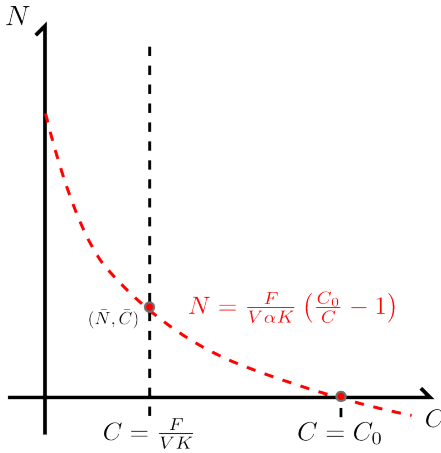


Figure 1: The $\dot{C} = 0$ nullcline (dashed red) intersecting with the $\dot{N} = 0$ nullcline (dashed black). The trivial and non-trivial steady-states $(0, C_0)$ and (\bar{N}, \bar{C}) are shown as red dots.

By placing Eq. (3) inside Eq. (4), we can find the non-trivial steady-state, the intersection of null-clines:

$$\begin{aligned}\bar{N}(\bar{C}) &= \frac{FC_0}{V\alpha K\bar{C}} - \frac{F}{V\alpha K} \\ &= \frac{FC_0}{V\alpha K \frac{F}{VK}} - \frac{F}{V\alpha K} \\ &= \frac{C_0}{\alpha} - \frac{F}{V\alpha K} = \frac{1}{\alpha} \left(C_0 - \frac{F}{VK} \right).\end{aligned}\tag{5}$$

The unknown parameters in Eq (5) are α and K . We can find these parameters by fitting Eq (5) to the data by non-linear least squares fitting \bar{N}_i at F_i for $i = 1, \dots, n$, where n is the number of observations.

After we obtain estimates of these parameters for both the killer yeast L and sensitive yeast S , (α_L , α_S and K_L , K_S respectively), we can model a “what if” scenario whereby we place both species of yeast, sensitive and killer, into one chemostat. The population of sensitive yeast S will be negatively impacted by the amount of toxin the killer yeast K can produce, so we add the term $-\beta KL$ to the differential equation describing population S . The differential equations we use to solve this three-species model is

$$\frac{dL}{dt} = K_L CL - \frac{FL}{V}, \quad (6)$$

$$\frac{dS}{dt} = K_S CS - \frac{FS}{V} - \beta SL, \quad (7)$$

$$\frac{dC}{dt} = -\alpha_L K_L CL - \alpha_S K_S CS - \frac{FC}{V} + \frac{FC_0}{V}. \quad (8)$$

The data we are provided with include two sets of two separate runs, along with the concentration of nutrient in the reservoir, $C_0 = 0.02$:

1 K1 Run

Vessel One :

Volumes/Hr	0.028	0.099	0.142	0.207	0.269	0.287	0.352	0.403
Optical Density at Steady State	0.144	0.151	0.099	0.069	0.045	0.02	0.003	0

Vessel Two :

Volumes/Hr	0.054	0.11	0.141	0.199	0.257	0.296	0.348	0.397	0.41
Optical Density at Steady State	0.164	0.151	0.11	0.092	0.072	0.023	0.006	0.002	0.004

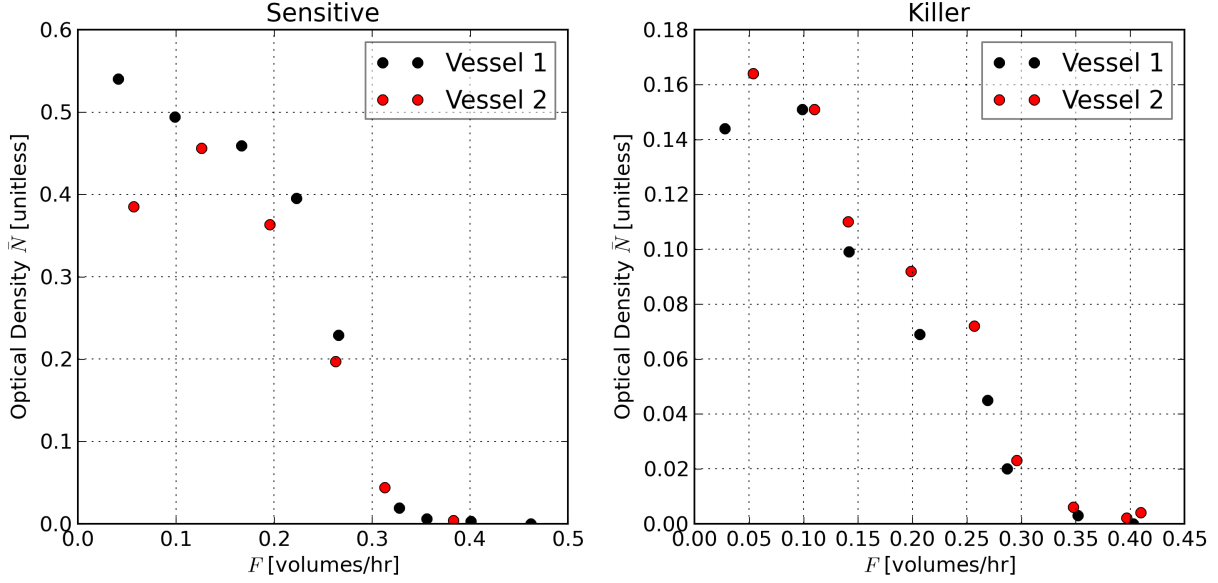
2 Sensitive Run

Vessel One :

Volumes/Hr	0.041	0.099	0.167	0.223	0.266	0.328	0.356	0.401	0.462
Optical Density at Steady State	0.54	0.494	0.459	0.395	0.229	0.019	0.006	0.003	0

Vessel Two :

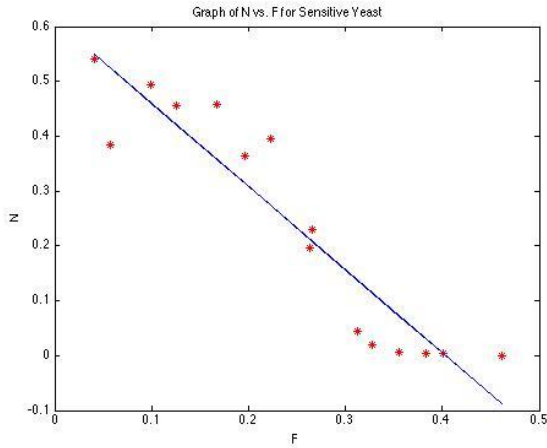
Volumes/Hr	0.0571	0.126	0.196	0.263	0.313	0.383
Optical Density at Steady State	0.385	0.456	0.363	0.197	0.044	0.004



The data provided does not include the volume of the chamber, V . In order to solve Eq. (5), this quantity is needed. Here we have dimensional analysis of the problem:

$$\begin{aligned}
 \frac{dN}{dt} &= KCN - \frac{FN}{V} \\
 &\equiv \left[\frac{1}{\text{time}} \right] \equiv \left[\frac{1}{\text{time}} - \frac{\text{volume}}{\text{time}} \cdot \frac{1}{\text{volume}} \right] \equiv \left[\frac{1}{\text{time}} \right], \\
 \frac{dC}{dt} &= -\alpha KCN - \frac{FC}{V} + \frac{FC_0}{V} \\
 &\equiv \left[\frac{1}{\text{time}} \right] \equiv \left[-\frac{1}{\text{time}} - \frac{\text{volume}}{\text{time}} \cdot \frac{1}{\text{volume}} + \frac{\text{volume}}{\text{time}} \cdot \frac{1}{\text{volume}} \right] \equiv \left[\frac{1}{\text{time}} \right].
 \end{aligned}$$

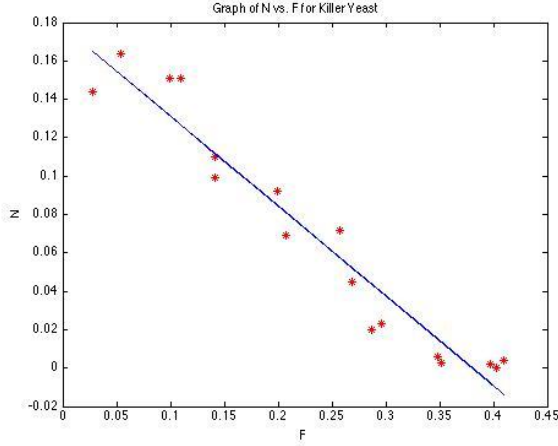
Taking $V = 1$, we can solve Eq. (5) using MatLab's `nlinfit` function. The results of this are shown below for both sensitive and killer yeast.



	Estimates	SE	CI
α_S	0.0325	0.0024	(0.0255, 0.0399)
K_S	20.1818	1.0802	(16.9281, 23.4356)

The results on the left show that there may be a better fit to the data than Eq. (5). Notice that elimination of the last few data points corresponding to high F may be removed; this would reduce the standard error and hence provide a better estimate for α_L and K_L .

Figure 2: Sensitive yeast S steady-state best-fit line using Eq. (5).



	Estimates	SE	CI
α_L	0.1124	0.0053	(0.0969, 0.1279)
K_L	19.0288	0.6367	(17.1526, 20.905)

The best-fit line on the left shows that steady-state data follows a fairly linear relationship with flow, and as such we can be confident our estimates for α_L and K_L are correct.

Figure 3: Killer yeast L steady-state best-fit line using Eq. (5).

Now that we have estimates for all values of α and K (for both killer and sensitive yeast) we can run the dynamic model (6), (7), and (8) to equilibrium. By letting β range from 0 to 0.5 (in units 1/time), and F range from 0 to 0.5 (in units volume/time), we can run the model for each β and F to determine the regions in the β, F plane where sensitive yeast S overtakes the killer yeast L and vice versa.

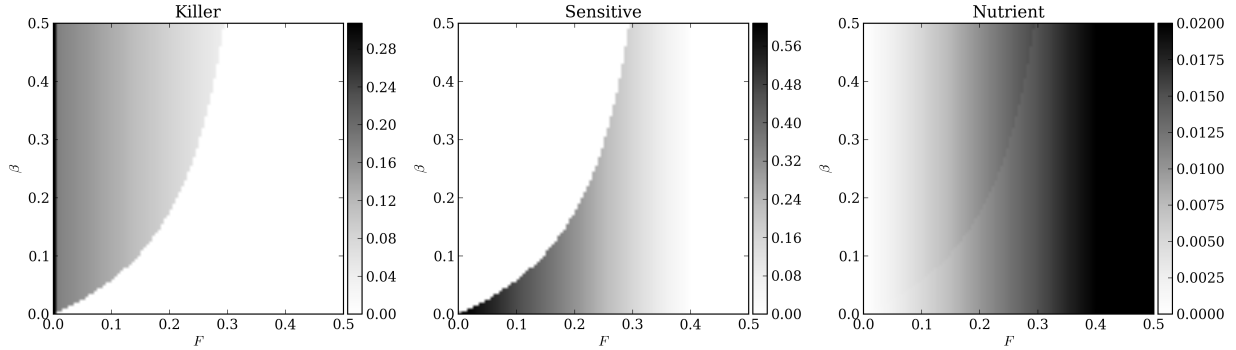
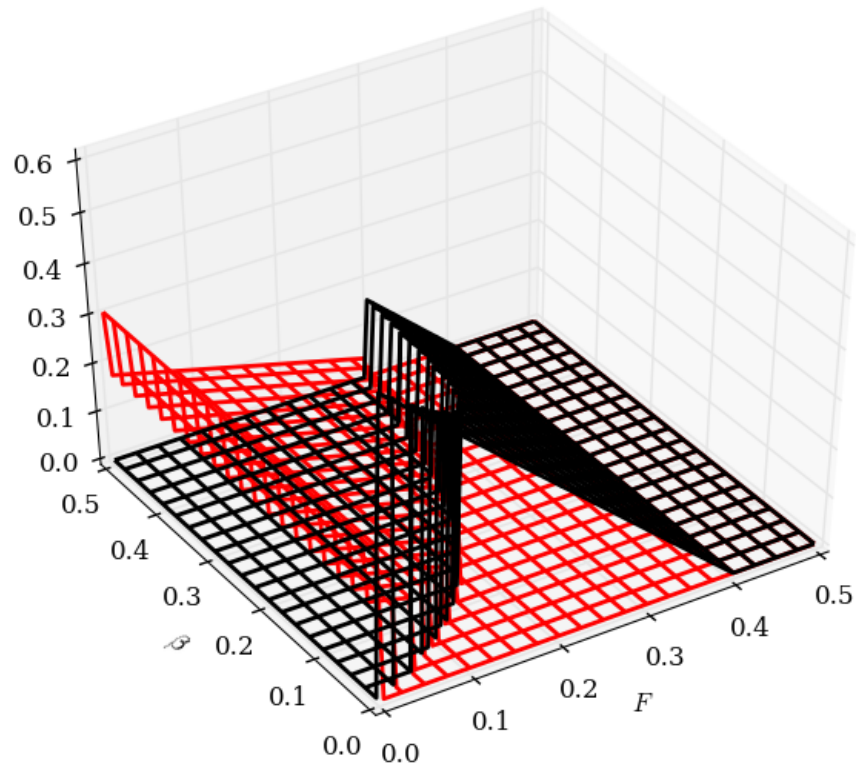


Figure 4: 250×250 steady-state optical density solution for killer yeast (left), sensitive yeast (middle) and nutrient (right) for a total run time of 40,000 hours. Equations (6), (7), and (8) were solved with the Dormand-Prince numerical integration algorithm with an absolute tolerance of $1e-6$, relative tolerance of $1e-6$, and timestep Δt of 500 hours. The timestep was kept high due to the $250 \times 250 = 62,500$ simulations required to complete the figure. In order to keep the total time to complete the simulation minimized, parallel processing was required.

Notice in Figure 4 that as β increases the killer yeast dominates and sensitive yeast eventually dies off, while as F increases the sensitive yeast dominates and the killer yeast eventually dies out. The sensitive yeast are flushed from the container at around $F = 0.4$ volumes/hr.



Source Code:

```

from scipy.integrate._ode      import ode
from scipy.io                 import savemat
from pylab                   import *
from mpl_toolkits.mplot3d     import Axes3D
from multiprocessing          import Queue, cpu_count, Process
from mpl_toolkits.axes_grid1  import make_axes_locatable

#-----
# ODE function to be integrated
def dLdt(L, S, C, params):
    """
    INPUT:
        L - population of killer yeast.
        S - population of sensitive yeast.
        C - population of nutrient.
        params - dLdt equation parameters.
    OUTPUT:
        dLdt - time derivative of L.
    """
    K_L = params[0]
    F    = params[1]
    V    = params[2]
    dLdt = K_L*C*L - F*L/V
    return array(dLdt)

```

```

def dSdt(L, S, C, params):
    """
    INPUT:
        L - population of killer yeast.
        S - population of sensitive yeast.
        C - population of nutrient.
        params - dSdt equation parameters.
    OUTPUT:
        dSdt - time derivative of S.
    """
    K_S = params[0]
    F = params[1]
    V = params[2]
    beta = params[3]
    dSdt = K_S*C*S - F*S/V - beta*S*L
    return array(dSdt)

def dCdt(L, S, C, params):
    """
    INPUT:
        L - population of killer yeast.
        S - population of sensitive yeast.
        C - population of nutrient.
        params - dCdt equation parameters.
    OUTPUT:
        dCdt - time derivative of C.
    """
    a_L = params[0]
    a_S = params[1]
    K_L = params[2]
    K_S = params[3]
    F = params[4]
    V = params[5]
    C_0 = params[6]
    dCdt = -a_L*K_L*C*L - a_S*K_S*C*S - F*C/V + F*C_0/V
    return array(dCdt)

def f(t, y, dLdt, dSdt, dCdt, L_params, S_params, C_params):
    """
    INPUT:
        t - time array
        dLdt - function
        uSdt - function
        dCdt - function
        L_params - parameters for dLdt
        S_params - parameters for dSdt
        C_params - parameters for dCdt
    OUTPUT:
        ydot[0] = time derivative of y[0],
        ydot[1] = time derivative of y[1],
        ydot[2] = time derivative of y[2].
    """
    L = y[0]
    S = y[1]
    C = y[2]
    rhs1 = dLdt(L, S, C, L_params) # right hand side 1st eqn
    rhs2 = dSdt(L, S, C, S_params) # right hand side 2nd eqn
    rhs3 = dCdt(L, S, C, C_params) # right hand side 3rd eqn
    return array([rhs1, rhs2, rhs3])

def model(F, beta, y0, ta, dt):
    """
    Run model for given volume flow rate <F> and toxin coef <beta> for total
    time array <ta> in hours at timestep <dt>, also in hours. Returns the
    last solution 3-tuple for L, S, and C.
    """
    # Additional parameters being passed to the ODE function
    a_L = 0.1124

```

```

a_S = 0.0325
K_L = 19.0288
K_S = 20.1818
V = 1.0
C_0 = 0.02

L_params = [K_L, F, V]
S_params = [K_S, F, V, beta]
C_params = [a_L, a_S, K_L, K_S, F, V, C_0]

# Call function that integrates the ODE:
r = ode(f)
r.set_integrator('dopri5', atol=1e-6, rtol=1e-5)
r.set_initial_value(y0, ta)
r.set_f_params(dLdt, dSdt, dCdt, L_params, S_params, C_params)

sol = []
sol.append(y0)
for t in ta[:-1]:
    r.integrate(r.t + dt)
    sol.append(r.y)
sol = array(sol).T

return sol[:, -1]

class solveProcess(Process):
    """
    Process to solve the model function.
    """
    def __init__(self, i, queue, beta_a, F_a, y0, ta, dt, p):
        """
        Initialize the Process with ID <i>, processing queue <queue>, beta array
        <beta_a>, flow array <F_a>, time array <ta>, timestep <dt>, and number of
        parameters <p>.
        """
        Process.__init__(self)
        self.i = i
        self.q = queue
        self.beta_a = beta_a
        self.F_a = F_a
        self.y0 = y0
        self.ta = ta
        self.dt = dt
        self.m = len(beta_a)
        self.n = len(F_a)
        self.p = p

    def run(self):
        """
        solve the differential equations for all beta_a and F_a.
        """
        p = self.p
        m = self.m
        n = self.n

        SS_L = zeros((m,n))
        SS_S = zeros((m,n))
        SS_C = zeros((m,n))

        for i, beta in enumerate(self.beta_a):
            for j, F in enumerate(self.F_a):
                print 'Process %i solving: beta=%f, F=%f' % (self.i, beta, F)
                sol = model(F=F, beta=beta, y0=self.y0, ta=self.ta, dt=self.dt)
                SS_L[i,j] = sol[0]
                SS_S[i,j] = sol[1]
                SS_C[i,j] = sol[2]

```



```

        self.q.put(array([SS_L, SS_S, SS_C])) # add the result to the queue.

def plot_sol(ax, f, extent, tit, cmap='Greys'):
    """
    plot the 2D solution <f> to axes <ax>.
    """
    im = ax.imshow(f[:,-1,:], extent=extent, cmap=cmap)
    divider = make_axes_locatable(ax)
    cax = divider.append_axes("right", size="5%", pad=0.05)
    ax.set_title(tit)
    ax.set_ylabel(r'$\beta$')
    ax.set_xlabel(r'$F$')
    colorbar(im, cax=cax)

# parameters :
m = 250 # number of beta discretizations.
n = 250 # number of F discretizations.
p = 3 # number of parameters
t0 = 0.0 # initial time
tf = 40000 # final time
dt = 500 # time step

# Initial conditions
y0 = [0.3, 0.3, 0.001]

# range of beta, flow, and time to model :
betaMin = 0.0
betaMax = 1.0
Fmin = 0.0
Fmax = 0.5

beta_a = linspace(betaMin, betaMax, m)
F_a = linspace(Fmin, Fmax, n)
ta = arange(t0, tf+dt, dt)

# multiprocessing data structures :
solvers = []
queue = []
numCpus = cpu_count()
Fs = array_split(F_a, numCpus)

# create a solver for each processor and begin solving each :
for i in range(numCpus):
    q = Queue()
    queue.append(q)
    solver = solveProcess(i, q, beta_a, Fs[i], y0, ta, dt, p)
    solvers.append(solver)
    solver.start()

# wait until solver (started above) finishes :
for s in solvers:
    s.join()

# retrieve the results :
sols = []
for q in queue:
    while q.empty() == False:
        sols.append(q.get())

# put the results from the individual cores back together :
for i, s in enumerate(sols):
    if i == 0:
        L_sol = s[0]
        S_sol = s[1]
        C_sol = s[2]
    else:
        L_sol = hstack((L_sol, s[0]))
        S_sol = hstack((S_sol, s[1]))

```

```

C_sol = hstack((C_sol, s[2]))

Beta, Flow = meshgrid(beta_a, F_a)

data = {'Beta' : Beta,
        'Flow' : Flow,
        'L_sol' : L_sol,
        'S_sol' : S_sol,
        'C_sol' : C_sol}
savemat('.../killer_yeast/data/results.mat', data)

# plot the results :
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.plot_wireframe(Beta, Flow, L_sol, color='r', lw=2.0, rstride=5, cstride=5)
ax.plot_wireframe(Beta, Flow, S_sol, color='k', lw=2.0, rstride=5, cstride=5)
ax.set_ylabel(r'$\beta$')
ax.set_xlabel(r'$F$')
show()

fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(131)
ax2 = fig.add_subplot(132)
ax3 = fig.add_subplot(133)

extent = [betaMin, betaMax, Fmin, Fmax]
plot_sol(ax1, L_sol, extent, r'Killer', cmap='Greys')
plot_sol(ax2, S_sol, extent, r'Sensitive', cmap='Greys')
plot_sol(ax3, C_sol, extent, r'Nutrient', cmap='Greys')
tight_layout()
savefig('.../killer_yeast/doc/images/sols.png', dpi=300)
show()

```