Mobile Toxins May Payoff

George Shillcock

george.shillcock@linacre.ox.ac.uk

Oxford Interdisciplinary Bioscience DTP UKRI-BBSRC

Abstract

The ecology of *E. coli* competition is discussed, particularly the implications of mobile toxins and niche overlap on invasion dynamics. I propose a system of differential equations to model and recapitulate previous experimental work. As well as present a sensitivity analysis which supports the main result: conjugation of toxin plasmids can in principle provide a competitive and evolutionary advantage to donors, despite rendering those same toxins ineffective.

Keywords: Microbial Competition, Plasmid, Conjugation, Toxin

Software Availability Scripts and data files are available from the companion repository.

Introduction

The world of microbiology is a myriad place. Microbiotic competitions are of significant importance to the health of humans and their symbiotic microbiota, in the advancement of therapeutics, as well as in agriculture. In this paper, I will explore intra-specific bacterial competition between strains of *Escherichia coli*, and how the outcomes of such competitions are influenced by niche overlap, toxins, and horizontal gene transfer. To do so I will take a theoretical approach, building on previous mathematical models of *E. coli* ecology.[1–3] Analysis of the model will allow us to shed light on the interactions between these processes and determine the significance of parameters to competition outcomes - with the scope to inform experimental design in the future. Much of this project was done in conjunction with the experimental work of Elisa Granato, and so, many specificities of this analysis align with her work.

In order to capture the kinetics that *E. coli* might face in a natural habitat such as the human gut, or indeed an *in vitro* experiment, our model considers two types of competition. Firstly, strains may compete passively by consuming a nutrient that they share. To succinctly capture nutrient diversity and niche overlap, I consider a simplified scenario in which there are only two nutrients available: glucose, which is communal to all *E. coli*, and sorbitol, which only some strains may consume. Secondly, strains may actively interfere with one another's growth by producing toxins. Bacteriocins are proteins that are produced by various species of bacteria, which inhibit the growth of related strains. In particular, colicins are a type of bacteriocin which are toxic to, and produced by, *E. coli* and other Enterobacteriacae. Colicins bind on the surface of sensitive cells and inhibit growth by forming pores in the cell membrane, acting as nucleases in the cytosol, or degrading the peptidoglycan layer.[4] Importantly, owing to the surface receptors, colicins are often active against closely related niche competitors, making them important determinants of intraspecific competition outcomes.[5] They may empower strains to invade established microbial communities, or alternatively, play a defensive role - prohibiting the advance of invading strains into an already-colonised niche.

An important feature of colicin toxins, and indeed bacteriocins generally, is that they are commonly found on plasmids.[6] These plasmids typically encode the toxin itself, a cognate immunity protein, and in some cases the toxin release mechanism, and are replicated independently from the chromosome. In addition, through conjugation, plasmids can transfer horizontally to nearby cells of different strains, as shown in Figure 1.

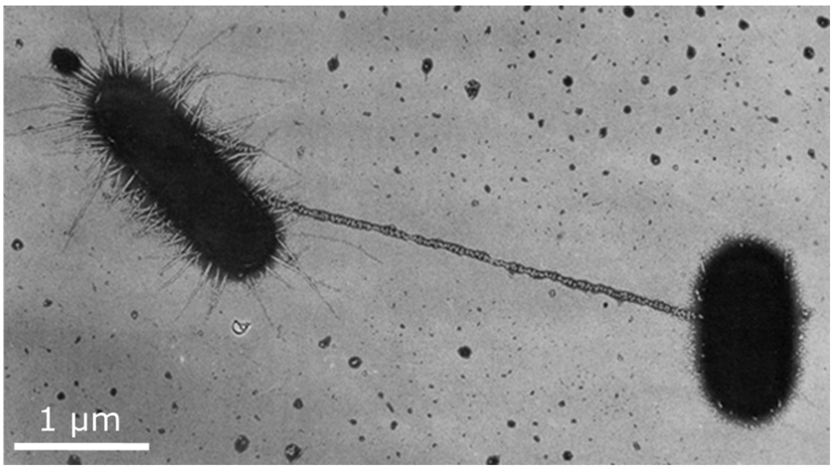


Figure 1: Conjugation between two *E. coli* cells. Modified from 7.

Of all known plasmids, around half are either conjugative or mobilisable,[8] meaning they either transfer autonomously between cells or exploit the transfer machinery encoded by other plasmids.[9] One consequence of transferring a toxin plasmid to nearby cells of a different strain is that genes which encode for toxin immunity are passed to potential competitors, thereby lowering their effectiveness. Though conjugation of toxin plasmids between strains is a rare occurrence, its effect may be significant. Preliminary experiments, by Granato, have revealed that conjugation persists despite toxins rapidly depleting the recipient population.

The question then arises: why are toxin plasmids in *E. coli* commonly found on mobilisable and conjugative? One proposal is that bacteriocin-encoding plasmids may act as parasites, whose evolutionary interests are not necessarily aligned with those of their bacterial hosts.[10] Much like plasmid-encoded toxin-antitoxin systems, toxin plasmids can promote their own persistence in a population by promoting the killing of plasmid-free daughter cells.[11] Plasmid control over their own transmission and selection within a population might therefore explain why *E. coli* endure mobile weapons. One alternative that has, to my knowledge, received less attention is whether donors ever gain a competitive advantage from conjugating, despite their weapon becoming ineffective.

Model

In order to explore this question further, I propose a simplistic theoretical model depicted in Figure 2. Modelling in this way enables us to determine how ecological processes interact and influence the outcome of bacterial competition.

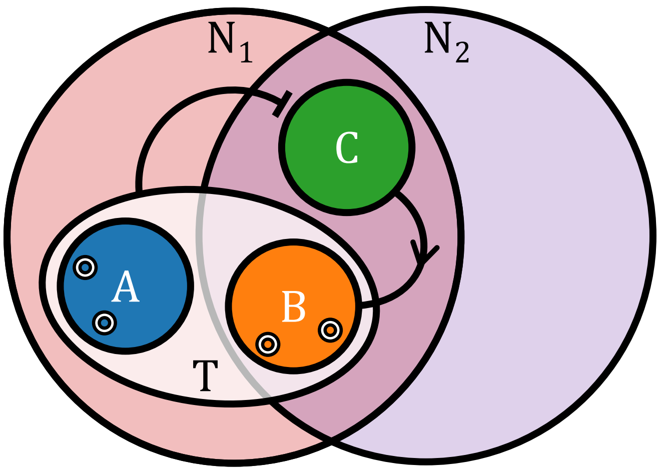


Figure 2: Donors (A), transconjugants (B) and recipients (C) compete via colicin toxin (T) over glucose (N_1) and sorbitol (N_2).

This model builds on previous work by using differential equations (1) to track the abundance of three *E. coli* strains over time.[1-3,12] Donors, who have concentration A, contain a conjugative (donable) toxin plasmid. The production of toxins is costly, due in part to cell lysis during release, and so diminishes the maximum attainable growth rate. The growth of donors is also limited by a saturating function of a communal nutrient N_1.[13]

$$\frac{dA}{dt} = (1 - f_A)\, r \frac{N_1}{N_1 + K_1} A \tag{1a}$$

Recipients, C, are receptible to plasmids, sensitive to the toxin, and able to consume an additional nutrient N_2. Conjugation is modelled as a second-order reaction between the concentrations of plasmid-bearers and recipients.[14]

$$\frac{dC}{dt} = r\left(p\frac{N_1}{N_1 + K_1} + \frac{N_2}{N_2 + K_2}\right) C - k\frac{T}{T + K_T} C - b(A + B)C \tag{1b}$$

Recipients which successfully receive a plasmid are relabelled as transconjugants, and have concentration B. Transconjugants consume both nutrients and, since they bear a plasmid, produce costly toxins and conjugate according to mass action.

$$\frac{dB}{dt} = (1 - f_B)\, r\left(p\frac{N_1}{N_1 + K_1} + \frac{N_2}{N_2 + K_2}\right) B + b(A + B)C \tag{1c}$$

The nutrient and toxin concentrations are also tracked explicitly. Niche overlap may be increased by a parameter p, which determines the degree to which recipients and transconjugants access the communal nutrient N_1.

$$\frac{dN_1}{dt} = -\frac{N_1}{N_1 + K_1}\left(A + p\,(B + C)\right) \tag{1d}$$

$$\frac{dN_2}{dt} = -\frac{N_2}{N_2 + K_2}\,(B + C) \tag{1e}$$

Toxin production rate is proportional to donor and transconjugant abundance, subject to their nutrient uptake. Toxins are assumed to be inert on the timescale of lysis/absorption and are only used up by killing.

$$\frac{dT}{dt} = f_A \mu \frac{N_1}{N_1 + K_1} A + f_B \mu \left(p\frac{N_1}{N_1 + K_1} + \frac{N_2}{N_2 + K_2}\right) B - \frac{T}{T + K_T} C \tag{1f}$$

It is perhaps important to note an alternative way to model niche overlap. In the model presented, changing p only affects the degree to which strains B and C access communal glucose, and has no effect on their ability to utilise their additional nutrient, N_2. In this way, we may describe p as capturing *metabolic diversification*. An alternative, however, would be to treat p as the proportion of the total effort a strain invests in feeding - instead capturing *metabolic diversion*. The difference being that turning attention towards communal glucose is at the expense of sorbitol. While we do not have time to test this distinction here, future work could address this by appending a factor $(1-p)$ to each Monod term containing N_2. Nutrient diversion might be a more appropriate phenotype if one ever decided to use this model to explore the evolutionary consequence of conjugation.

Table 1: Parameters of the model along with their default value. The differential equation model (1) uses the default values unless otherwise stated. The sensitivity analysis assumes distributions with medians given by the default values. X denotes the shared units in which the concentration of cells, nutrients, and toxins are measured.

Parameter	Interpretation	Units	Default Value
b	conjugation rate	$X^{-1}t^{-1}$	1
$0 \leq f_A, f_B \leq 1$	toxin allocation of strain A, B	1	0.5
k	toxin kill rate	t^{-1}	10
K_T	half-saturation concentration of toxin	X	10^{-4}
K_1, K_2	half-saturation concentration of nutrient $1, 2$	X	1
$0 \leq p \leq 1$	niche overlap proportion	1	0.5
r	maximum growth rate	t^{-1}	1
μ	toxin secretion rate	t^{-1}	100

The next step of the analysis is to numerically solve system (1) - details of which are left to Appendix A. It is, however, important to emphasise that more work is required to ensure the robustness of the solver. Following this analysis, we obtain trajectories such as those shown in Figure 3. The figure is split into two rows, with each row showing the change in the concentration of the cells, nutrients and donors over time. The bottom row uses the default parameters from Table 1, which ensure there is conjugation, toxin production, and niche overlap. In the top row, the only difference is that I have ‘switched off’ conjugation, by setting $b = 0$.

This result allows us to answer the question posed affirmatively: there is a scenario in which

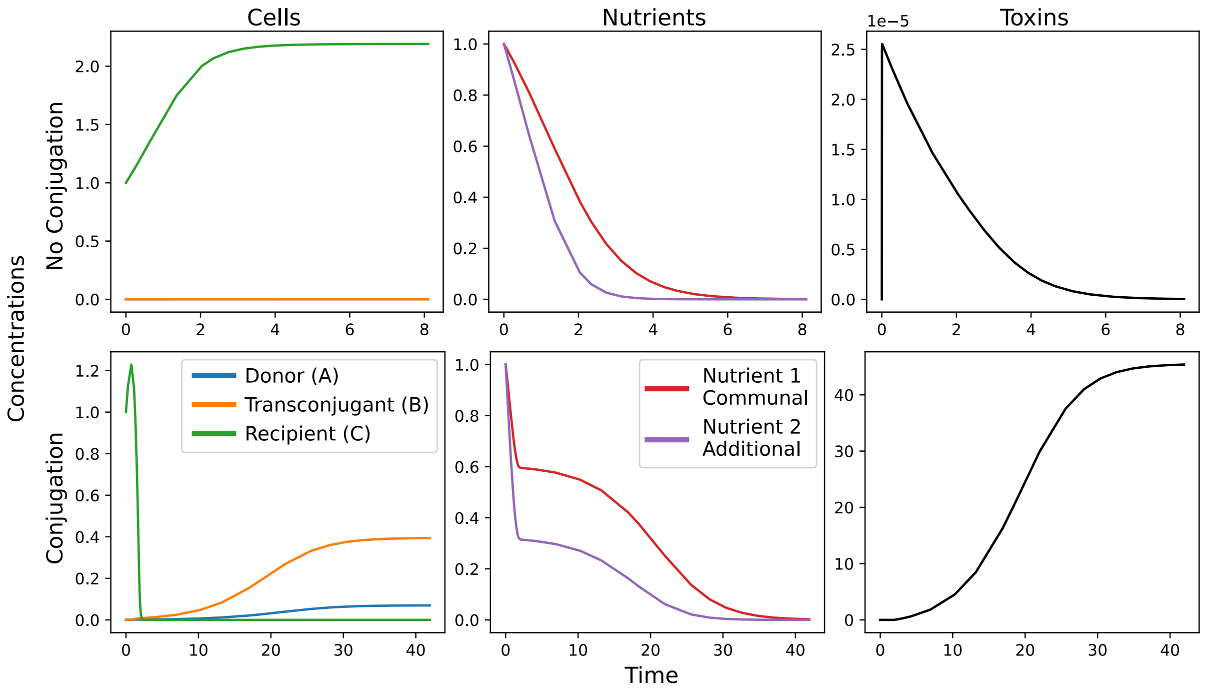


Figure 3: Two trajectories in which all model parameters are the same (default values from Table 1), except the top row has no conjugation ($b = 0$). The initial conditions used are $A = 10^{-3}, B = 0, C = 1, N_1 = 1, N_2 = 1, T = 0$.

conjugation has an ecological benefit to donor strain. If we consider a setup in which donors are rare and attempting to invade a niche, conjugation enables invasion by a combination of interference and exploitation competition. Firstly, the addition of conjugation enables donors to recruit transconjugants as toxin producers, thereby increasing the rate of toxin production. Secondly, transconjugants are able to utilise the second nutrient (sorbitol, N_2) to the detriment of the recipient population.

The combination of effects has, in this case, pushed the donors over a threshold past which they can invade from rare. Whilst the steady state of donors is low in absolute terms, and as compared to the abundance of transconjugants, when compared to the no-conjugation case the outcome is decisively improved for donors.

An important check is whether this result depends on the very specific choice of parameters used, or whether it is in some sense common. More generally, one might question which parameters significantly affect the model's outputs - in this case the steady-state abundances. To shed light on this in the next section I will present a sensitivity analysis.

Sensitivity Analysis

Sensitivity analysis is used to allocate uncertainty/variance in the output of a process to different sources of uncertainty/variance in the input. In general, the process under consideration could be a simulation, an analytical model, or even experimental trials. For a review see 15. The method of analysis is strongly problem-specific, and incorrect use is still rife in major publications across disciplines.[16] In this paper, we employ a method introduced by Sobol[17] and later influenced by Saltelli et al.,[18] implemented using the Python library SALib.[19]

Sensitivity Analysis Procedure

1. Propose distributions for each parameter of interest.
2. Sample from the joint distribution of the parameters.
3. Use these parameters to obtain a model output.
4. Repeat steps (1) and (2) many times.
5. Determine the uncertainty (variance) of the outputs.
6. Compute the contribution of each parameter to the uncertainty.

For each model parameter, one proposes a distribution which captures the possible breadth of parameter space (Figure 4). Certain properties of each parameter, for example, p being a proportion, constrain the reasonable possibilities of each distribution. However, choosing prior distributions does remain a subjective choice, and one to which the outcome of a sensitivity analysis is itself sensitive. Experimental support can help estimate parameters and this could be a direction for future work. The computational expense of the model is an important consideration in step (4). The number of model evaluations scales with the desired number of outputs and the number of samples taken. Therefore, one faces a trade-off between long run-time and accuracy.

Step (5) allows us to address the generality of the previous finding (Figure 3). In order to determine conjugation's effect on *E. coli* competition, I will compare the output variance when either the conjugation rate is fixed ($b = 0$) or is sampled. All other parameters are sampled from the same distributions. Since there is no direct cost of conjugation in this model, setting the rate parameter b to zero is equivalent to the toxin genes being encoded on no-mobilisable plasmids, or perhaps chromosomally.

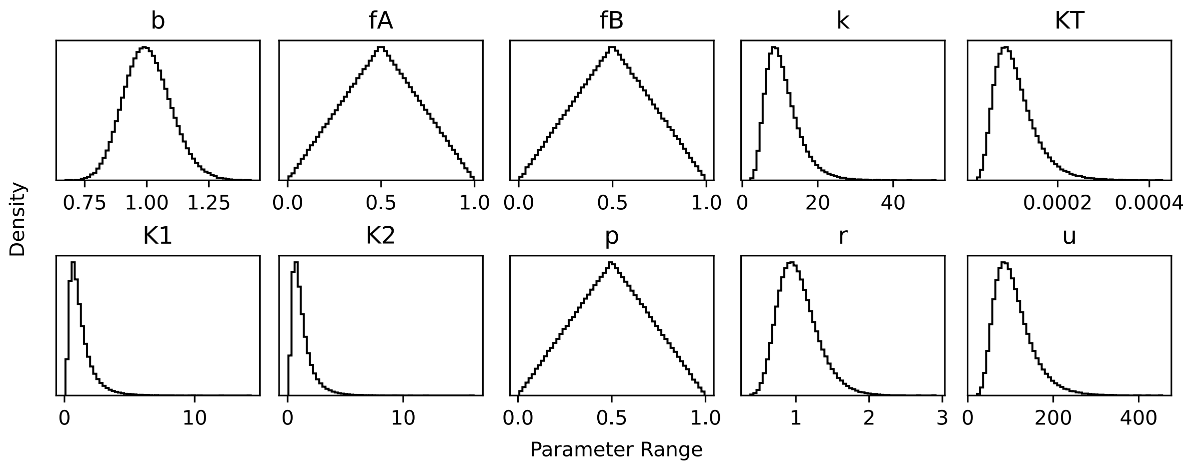


Figure 4: The sample distributions of each model parameter.

Figure 5 shows the distribution of the steady-state abundances, depending on whether conjugation is present or not. The conclusion is in agreement with the single trajectory results already presented. Conjugation enables a two-fold attack by donors which can result in their invasion into a niche.

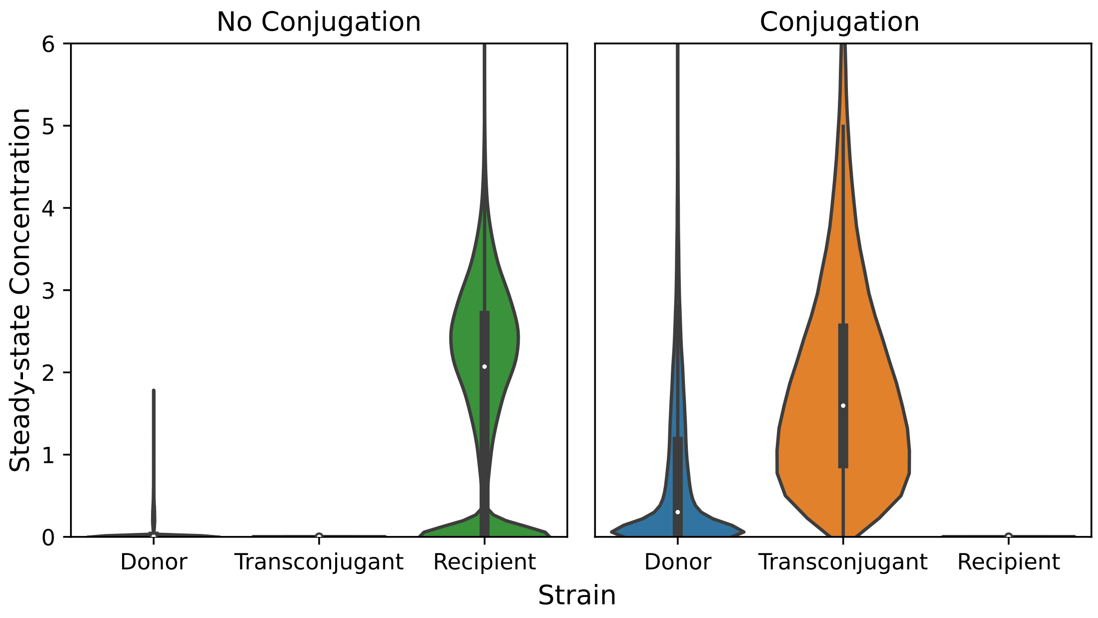


Figure 5: Each violin plot depicts the distribution of steady-state concentration for a given strain. In the left plot, a joint distribution in which the conjugation rate was fixed at zero ($b = 0$) is used, whereas, on the right, the conjugation rate is sampled from a lognormal distribution.

The final step in the analysis is to apportion the variance in the output to each parameter. A parameter's contribution to the variance is called its (total) Sobol index. It is calculated as the difference in the variance when each parameter is either fixed in the sample distribution or not. The SALib package also enables one to calculate the direct and pairwise contributions of each parameter, respectively called the first and second-order indices. Figure 6 plots these indices for both the conjugation and no-conjugation cases. I have only plotted the recipient indices in the no-conjugation case because these are the only population with any appreciable uncertainty (variance). Similarly, for the conjugation case, only donor and transconjugant indices are shown.

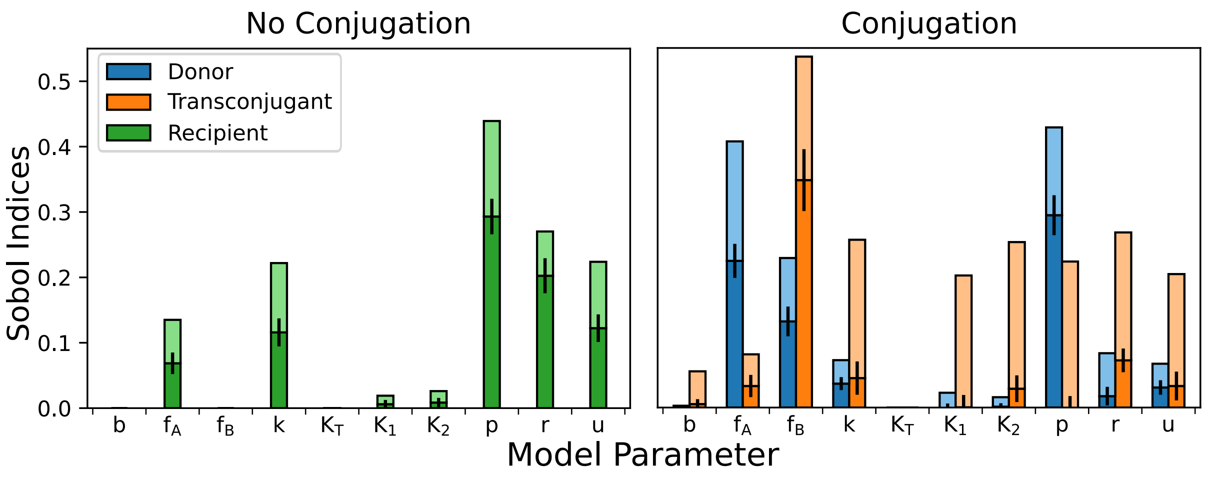


Figure 6: The Sobol sensitivity indices of each parameter. Within each bar, a parameter's contribution is split into its individual contribution (first-order) and the contribution owing to the interaction between parameters (higher-order). Thin vertical lines denote the confidence interval of the first-order index.

We can see from the figure that much of the sensitivity is due to interaction effects between the parameters, corresponding to the second-order indices. A more in-depth analysis could determine which parameters are cross-talking to produce this output variance. While the reliability of these measures is a topic for future work, the principle remains important. Being able to attribute sensitivity to each parameter increases our understanding of our model, and may – if the model is valid – help infer what experimental controls are most important.

Future Work

The main result of this paper highlights an antagonism between selection at the cell and patch levels. It would be interesting to address questions about how this system might evolve and to assign parameters as phenotypic traits under selection. Another avenue would be to alter the model output during the sensitivity analysis to address the different questions about the system. For example, one might be interested in the plasmid abundance, or indeed the cumulative abundance of toxin producers.

References

[1] V. Bucci, C. D. Nadell, and J. B. Xavier, “The evolution of bacteriocin production in bacterial biofilms,” *The American Naturalist*, vol. 178, no. 6, pp. E162–E173, 2011.

[2] J. D. Palmer and K. R. Foster, “The evolution of spectrum in antibiotics and bacteriocins,” *Proceedings of the National Academy of Sciences*, vol. 119, no. 38, p. e2205407119, 2022.

[3] R. Niehus, N. M. Oliveira, A. Li, A. G. Fletcher, and K. R. Foster, “The evolution of strategy in bacterial warfare via the regulation of bacteriocins and antibiotics,” *Elife*, vol. 10, p. e69756, 2021.

[4] E. Cascales, S. K. Buchanan, D. Duché, C. Kleanthous, R. Lloubès, K. Postle, M. Riley, S. Slatin, and D. Cavard, “Colicin biology,” *Microbiology and molecular biology reviews*, vol. 71, no. 1, pp. 158–229, 2007.

[5] L. P. Nedialkova, R. Denzler, M. B. Koeppel, M. Diehl, D. Ring, T. Wille, R. G. Gerlach, and B. Stecher, “Inflammation fuels colicin ib-dependent competition of salmonella serovar typhimurium and e. coli in enterobacterial blooms,” *PLoS pathogens*, vol. 10, no. 1, p. e1003844, 2014.

[6] M. A. Riley and J. E. Wertz, “Bacteriocins: evolution, ecology, and application,” *Annual Reviews in Microbiology*, vol. 56, no. 1, pp. 117–137, 2002.

[7] C. C. Brinton and L. Baron, “The properties of sex pili, the viral nature of “conjugal” genetic transfer systems, and some possible approaches to the control of bacterial drug resistance,” *CRC critical reviews in microbiology*, vol. 1, no. 1, pp. 105–160, 1971.

[8] C. Smillie, M. P. Garcillán-Barcia, M. V. Francia, E. P. Rocha, and F. de la Cruz, “Mobility of plasmids,” *Microbiology and Molecular Biology Reviews*, vol. 74, no. 3, pp. 434–452, 2010.

[9] A. E. Dewar, J. L. Thomas, T. W. Scott, G. Wild, A. S. Griffin, S. A. West, and M. Ghoul, “Plasmids do not consistently stabilize cooperation across bacteria but may promote broad pathogen host-range,” *Nature ecology & evolution*, vol. 5, no. 12, pp. 1624–1636, 2021.

[10] J. G. Lopez, M. S. Donia, and N. S. Wingreen, “Modeling the ecology of parasitic plasmids,” *The ISME Journal*, vol. 15, no. 10, pp. 2843–2852, 2021.

[11] R. F. Inglis, B. Bayramoglu, O. Gillor, and M. Ackermann, “The role of bacteriocins as selfish genetic elements,” *Biology letters*, vol. 9, no. 3, p. 20121173, 2013.

[12] J. C. R. Hernández-Beltrán, A. San Millán, A. Fuentes Hernández, and R. Peña-Miller, “Mathematical models of plasmid population dynamics,” *Frontiers in Microbiology*, p. 3389, 2021.

[13] C. D. Nadell, K. R. Foster, and J. B. Xavier, “Emergence of spatial structure in cell groups and the evolution of cooperation,” *PLoS computational biology*, vol. 6, no. 3, p. e1000716, 2010.

[14] Q. J. Leclerc, J. A. Lindsay, and G. M. Knight, “Mathematical modelling to study the horizontal transfer of antimicrobial resistance genes in bacteria: current state of the field and recommendations,” *Journal of the royal society interface*, vol. 16, no. 157, p. 20190260, 2019.

[15] B. Iooss and P. Lemaître, “A review on global sensitivity analysis methods,” *Uncertainty management in simulation-optimization of complex systems: algorithms and applications*, pp. 101–122, 2015.

[16] A. Saltelli, K. Aleksankina, W. Becker, P. Fennell, F. Ferretti, N. Holst, S. Li, and Q. Wu, “Why so many published sensitivity analyses are false: A systematic review of sensitivity analysis practices,” *Environmental modelling & software*, vol. 114, pp. 29–39, 2019.

[17] I. M. Sobol, “Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates,” *Mathematics and computers in simulation*, vol. 55, no. 1-3, pp. 271–280, 2001.

[18] A. Saltelli, P. Annoni, I. Azzini, F. Campolongo, M. Ratto, and S. Tarantola, “Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index,” *Computer physics communications*, vol. 181, no. 2, pp. 259–270, 2010.

[19] T. Iwanaga, W. Usher, and J. Herman, “Toward SALib 2.0: Advancing the accessibility and interpretability of global sensitivity analyses,” *Socio-Environmental Systems Modelling*, vol. 4, p. 18155, May 2022.

Code for Toxin Plasmids

March 31, 2023

1 Code for 'Toxin Plasmids Affect Ecological Competition'

[1]:
```python
import numpy as np

import matplotlib.pyplot as plt
from matplotlib.ticker import AutoMinorLocator
plt.rcParams.update({'mathtext.default': 'regular'})

import seaborn as sns

from scipy.integrate import solve_ivp
from scipy.stats import sem

import itertools as it

from SALib import ProblemSpec

from tqdm.notebook import tqdm

def m(X,k):
    '''
        Monod formular with half-saturation constant k.
        The maximum rate parameter is coded as a coefficient outside of the␣
 ↪function.
    '''
    return X/(X+k)


def HGT_TOXIN(time, variables, *parameters):
    '''
        Defines the change in abundance of a community of cells
        competing with toxins over nutrients.
    '''

    A, B, C, N1, N2, T = variables

    b, fA, fB, k, KT, K1, K2, p, r, u = parameters
    dAdt  = (1-fA)*r*m(N1,K1)*A

    dBdt  = (1-fB)*r*(p*m(N1,K1) + m(N2,K2))*B + b*(A + B)*C

    dCdt  = r*(p*m(N1,K1)+m(N2,K2))*C - k*m(T,KT)*C - b*(A + B)*C

    dN1dt = -m(N1,K1)*(A + p*(B + C))

    dN2dt = -m(N2,K2)*(B + C)

    dTdt  = fA*u*m(N1,K1)*A + fB*u*(p*m(N1,K1) + m(N2,K2))*B - m(T,KT)*C

    return dAdt, dBdt, dCdt, dN1dt, dN2dt, dTdt


def stop_condition(time, variables, *parameters):
    '''
        Integration is terminated when cell and nutrient concentrations have␣
 ↪stabilised.
    '''

    A, B, C, N1, N2, T = variables

    dAdt, dBdt, dCdt, dN1dt, dN2dt, _ = HGT_TOXIN(time, variables, *parameters)

    tolerance = 10**-3

    return abs(dAdt) + abs(dBdt) + abs(dCdt) + abs(dN1dt) + abs(dN2dt) -␣
 ↪tolerance


def integrate(model, ICs, params, maximum_time):
    '''
        Given a system of ODEs, returns the temporal dynamics.
    '''

    sol = solve_ivp(model, (0, maximum_time), list(ICs.values()),
                    args = list(params.values()),
                    events = stop_condition,
                    method = 'Radau' # opted for a stiff solver
                   )

    time = sol.t
    densities = np.transpose(sol.y)

    equilibriated = sol.status      # 0 if solver reached maximum_time
```

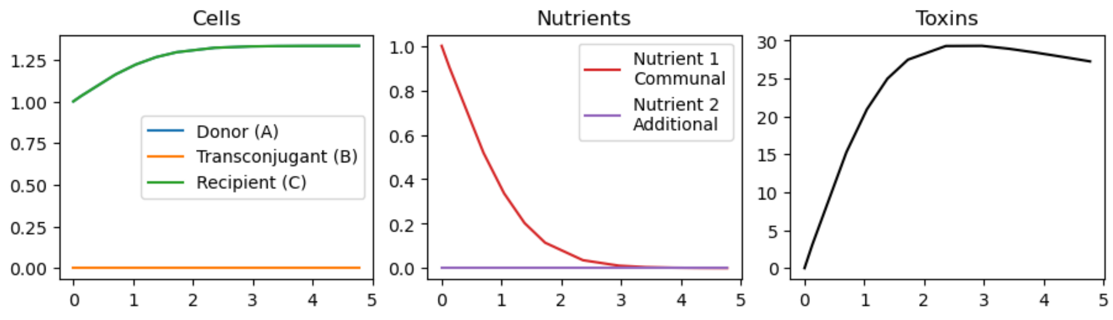
```
                                        # 1 if a termination event occurred

    ignore_stop = not stop_condition.terminal

    if equilibriated or ignore_stop:
        return time, densities

    else:
        raise Exception('Equilibrium not reached before maximum_time.')


def plot(time, densities, normalised=False, log_scaled=False):
    '''
        Plots the temporal dynamics, with options to
            (i) normalise the cell densities and/or
            (ii) plot on a log-scaled y-axis.
    '''

    cells = densities[:,0:3]
    nutrients = densities[:,3:5]
    toxin = densities[:,5]

    fig, ax = plt.subplots(1,3, figsize = (9,2.5), constrained_layout = True)

    if normalised:
        cells /=  cells.sum(axis=1)[:,np.newaxis]

    if log_scaled:
        ax[0].set_yscale('log')
        ax[1].set_yscale('log')
        ax[2].set_yscale('log')

    ax[0].set_title('Cells')
    ax[1].set_title('Nutrients')
    ax[2].set_title('Toxins')

    ax[0].plot(time, cells)
    ax[1].plot(time, nutrients[:,0],'#d62728')
    ax[1].plot(time, nutrients[:,1],'#9467bd')

    ax[2].plot(time, toxin, 'k')

    ax[0].legend(['Donor (A)', 'Transconjugant (B)', 'Recipient (C)'])
    ax[1].legend(['Nutrient 1 \nCommunal', 'Nutrient 2 \nAdditional'])

#     plt.savefig(f"tmp_figs/trajectories.svg", bbox_inches='tight')
    plt.show()
```

```
# Do not change the ORDER of the elements in the following dictionaries.

ICs = {'A' : 10**-3,
       'B' : 0,
       'C' : 1,
       'N1': 1,
       'N2': 1,
       'T' : 0
      }

params = {'b' : 1,
          'fA': .5,
          'fB': .5,
          'k' : 10,
          'KT': 10**-4,
          'K1': 1,
          'K2': 1,
          'p' : .5,
          'r' : 1,
          'u' : 100
         }

stop_condition.terminal = True # To ignore stop_condition and instead run to␣
 ↪maximum_time set to False
maximum_time = 2000
```

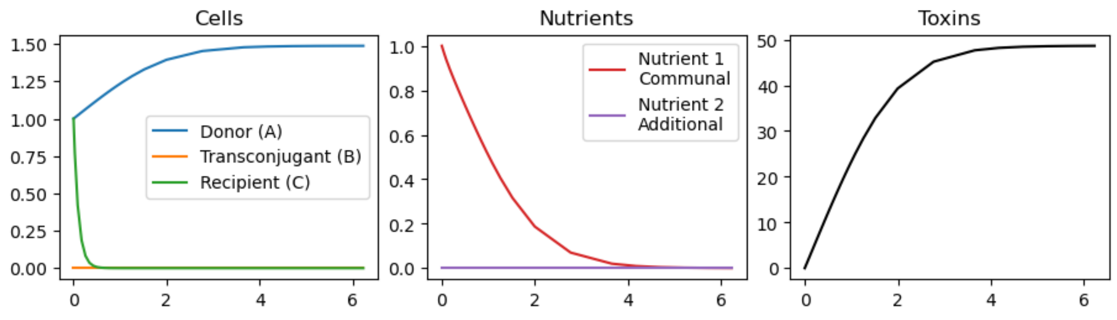
Some example outputs of the model are shown below.

[2]:
```
def run():
    print('Neither Toxin or Conjugation')
    params['b'] = 0
    params['k'] = 0
    time, densities = integrate(HGT_TOXIN, ICs, params, maximum_time)
    plot(time, densities)

    print('Toxin only')
    params['b'] = 0
    params['k'] = 10
    time, densities = integrate(HGT_TOXIN, ICs, params, maximum_time)
    plot(time, densities)

    print('Conjugation only')
    params['b'] = 1
    params['k'] = 0
    time, densities = integrate(HGT_TOXIN, ICs, params, maximum_time)
    plot(time, densities)
    print('Toxin and Conjugation')
    params['b'] = 1
    params['k'] = 10
    time, densities = integrate(HGT_TOXIN, ICs, params, maximum_time)
    plot(time, densities)

print('No Additional Nutrient and 1:1 Competition')
ICs['N2'] = 0
ICs['A'] = 1
ICs['C'] = 1

run()

print('Additional Nutrient and Invasion')
ICs['N2'] = 1
ICs['A'] = 10**-3
ICs['C'] = 1

run()
```

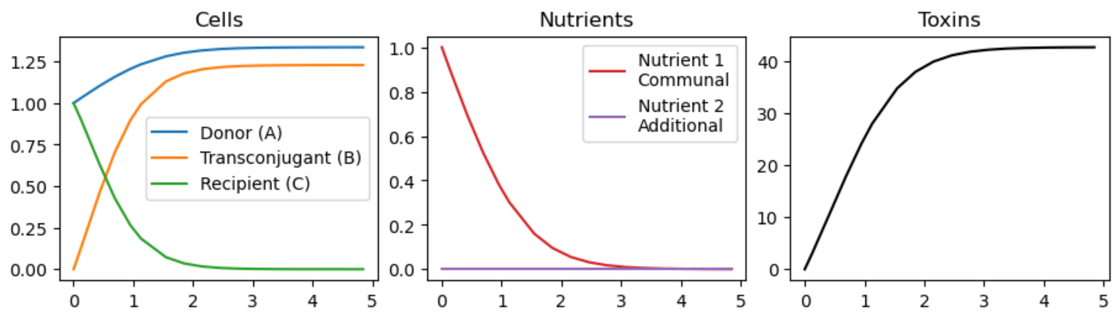
```
No Additional Nutrient and 1:1 Competition
Neither Toxin or Conjugation
```



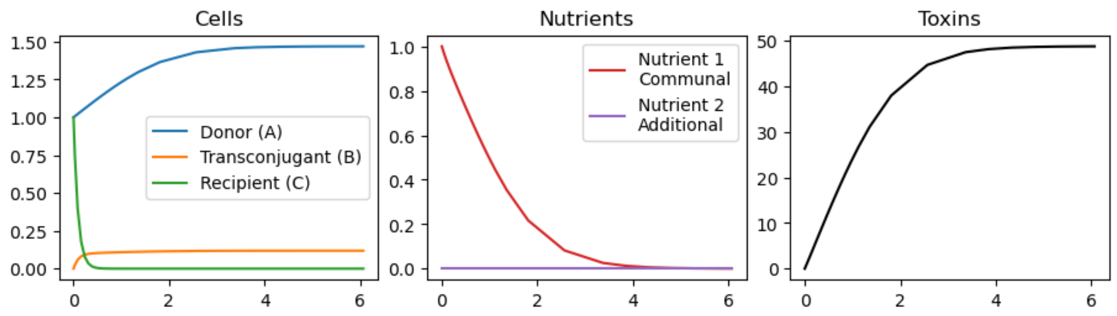
```
Toxin only
```



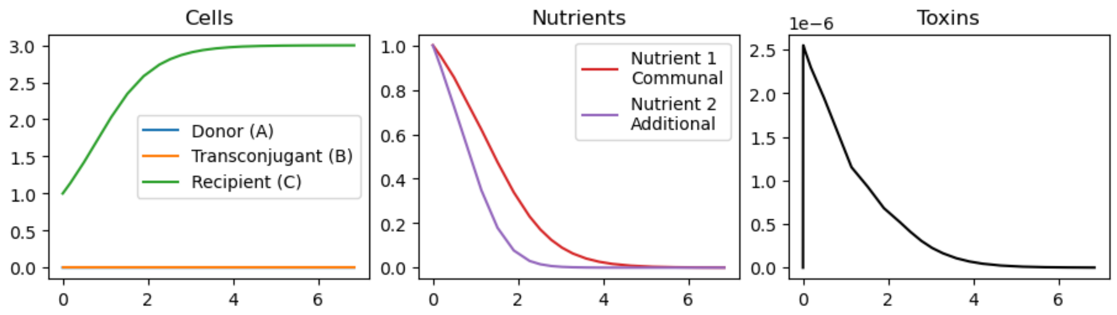
```
Conjugation only
```



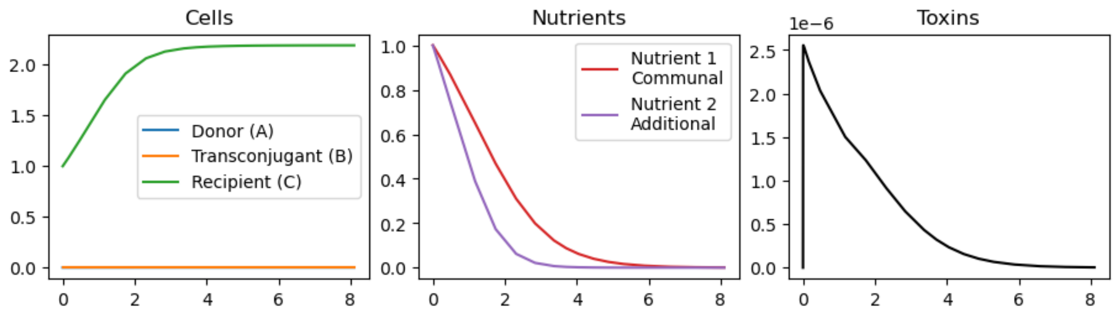
```
Toxin and Conjugation
```



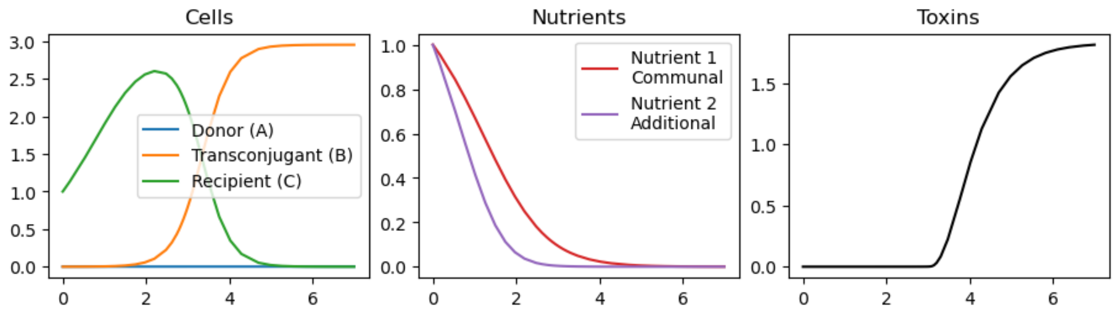
```
Additional Nutrient and Invasion
Neither Toxin or Conjugation
```



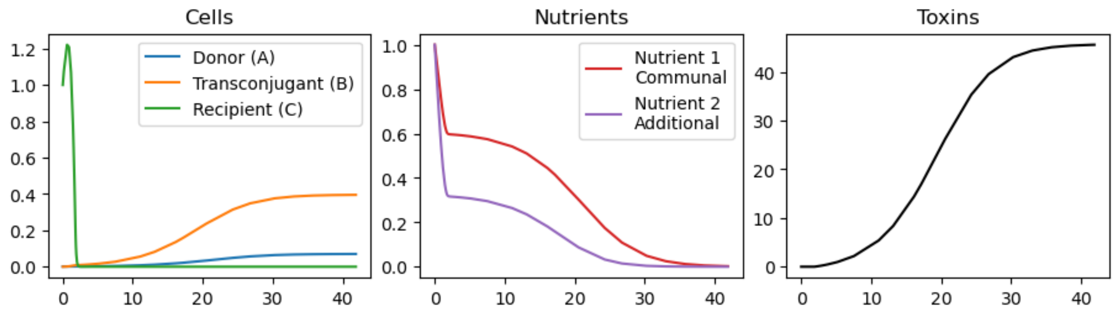
```
Toxin only
```



`Conjugation only`



`Toxin and Conjugation`



2 Sensitivity Analysis

Further reading SaLib docs: https://salib.readthedocs.io

SaLib tutorial: https://waterprogramming.wordpress.com/2016/02/25/salib-v0-7-1-group-sampling-nonuniform-distributions

Sensitivity Analysis Tutorial: https://uc-ebook.org/docs/html/A2_Jupyter_Notebooks.html

---

Firstly, we need to propose distributions for each parameter we intend to study. SaLib allows for triangular, normal, lognormal and uniform distributions. The distributions are each controlled by two parameters (called `'bounds'` but not always bounds):

- Triangular, `triang` (assumed lower bound of 0)
    1. width of distribution (scale, must be greater than 0)
    2. location of peak as a fraction of the scale (must be in [0,1])
- Normal, `norm`
    1. mean (location)
    2. standard deviation (scale, must be greater than 0)
- Lognormal(μ, σ^2), `lognorm` (natural logarithms, assumed lower bound of 0)
    1. ln-space mean (median/scale)
    2. ln-space standard deviation (>0) (shape), variance changes in same direction
- Uniform, `unif`
    1. lower bound
    2. upper bound (must be greater than lower bound)

All the analysis information (in/out samples, results) are stored in a `ProblemSpec` object that I've call `SA_spec`.

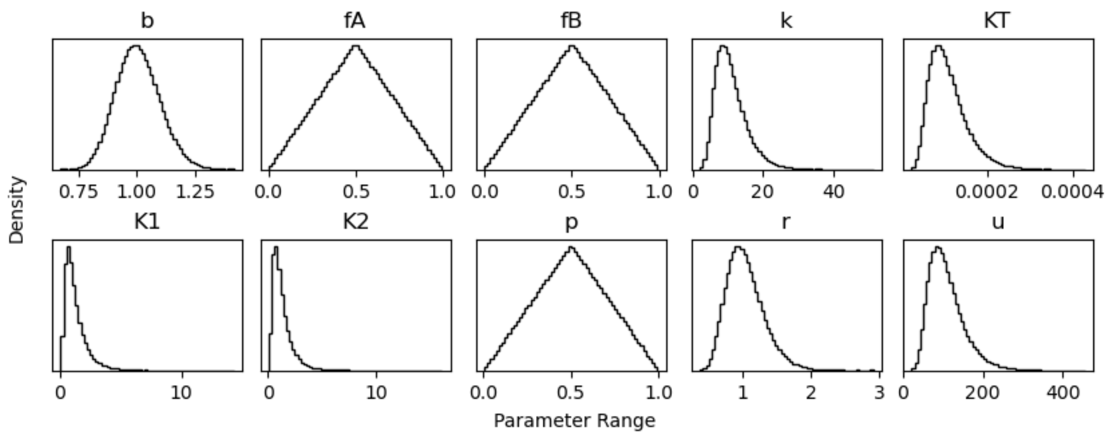
```
[3]: def setup(conjugation=False):

         param_distributions = {}

         if conjugation:
             param_distributions.update({'b' : ['lognorm', [np.log(1), np.log(1.
      ↪1)]]})
         else:
             params['b'] = 0

         param_distributions.update({'fA': ['triang',  [1, .5]],
                                     'fB': ['triang',  [1, .5]],
                                     'k' : ['lognorm', [np.log(10), np.log(1.5)]],
                                     'KT': ['lognorm', [np.log(10**-4), np.log(1.
      ↪5)]],
                                     'K1': ['lognorm', [np.log(1), np.log(2)]],
                                     'K2': ['lognorm', [np.log(1), np.log(2)]],
                                     'p' : ['triang',  [1, .5]],
                                     'r' : ['lognorm', [np.log(1), np.log(1.3)]],
                                     'u' : ['lognorm', [np.log(100), np.log(1.5)]]
                                     })

         return ProblemSpec({'num_vars': len(param_distributions),
                             'names'   : list(param_distributions.keys()),
                             'dists'   : [x[0] for x in param_distributions.
     ↪values()],
                             'bounds'  : [x[1] for x in param_distributions.
     ↪values()],
                             'outputs' : ['Donor', 'Transconjugant', 'Recipient']
                            })
```

We will visualise the proposed parameter distributions below...

Secondly, we sample from the joint probability distribution of our parameter space. With these samples we generate the corresponding steady-state cell densities, and save them to a file. Since this step may take a while feel free to use the file I made earlier by keeping `read_file = True`.

[4]:
```
read_in = True
conjugation = True

def plot_histograms(param_samples):

    fig, axes = plt.subplots(2,5, figsize=(8,3), layout="constrained")

    for i, ax in enumerate(axes.flat):
        try:
            ax.hist(param_samples[:,i], bins=60, density=True,␣
↪histtype='step',color='k')
            ax.set_title('$'+SA_spec['names'][i]+'$')
            ax.set_yticks([])
            ax.yaxis.set_tick_params(labelleft=False)

        except IndexError:
            pass

    fig.text(0.5, -.05, 'Parameter Range', ha='center')
    fig.text(-0.03, 0.5, 'Density', va='center', rotation='vertical')

    plt.savefig('tmp_figs/param_distributions.svg', bbox_inches='tight')
    plt.show()


if read_in:
    SA_spec = setup(conjugation)

    with open(f'conj_{conjugation}_sample_12.npy', 'rb') as f:
        param_samples = np.load(f)['samples']
        SA_spec.set_samples(np.load(f)['samples'])
        SA_spec.set_results(np.load(f)['results'])

    plot_histograms(param_samples)
```

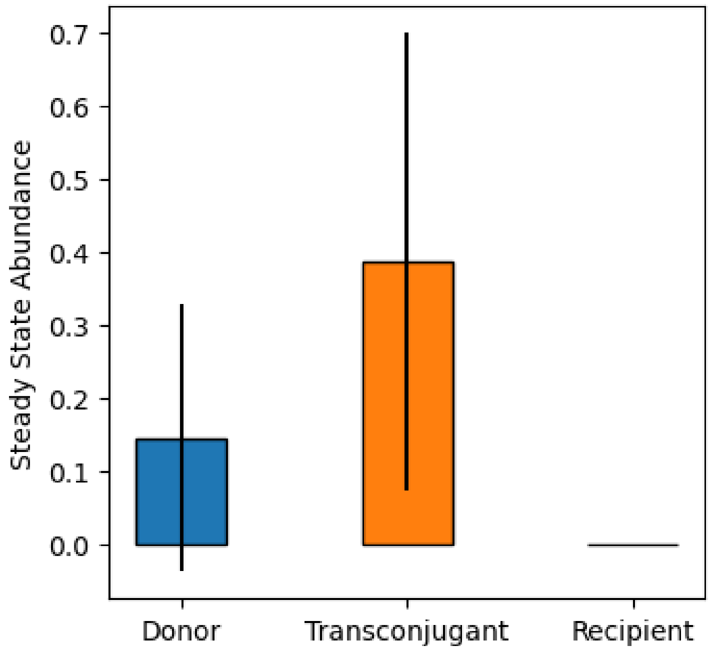
```
else:
    SA_spec = setup(conjugation)

    samples = 2**12 # default uses 2**12

    SA_spec.sample_sobol(samples, calc_second_order=True)

    param_samples = SA_spec.samples

    plot_histograms(param_samples)

    output_samples = np.zeros((np.shape(param_samples)[0],␣
 ↪len(SA_spec['outputs'])))

    for index, param_sample in enumerate(tqdm(param_samples)):

        params.update(dict(zip(SA_spec['names'], param_sample)))
        time, densities = integrate(HGT_TOXIN, ICs, params, maximum_time)
        output_samples[index,0:3] = densities[-1,0:3]

    SA_spec.set_results(output_samples)

    with open('HGT_TOXIN_new.npy', 'wb') as f:
        np.savez(f, samples=param_samples, results=output_samples)
```



We can visualise the uncertainty in the steady state abundances by plotting the mean and standard deviation of the sample model outputs.

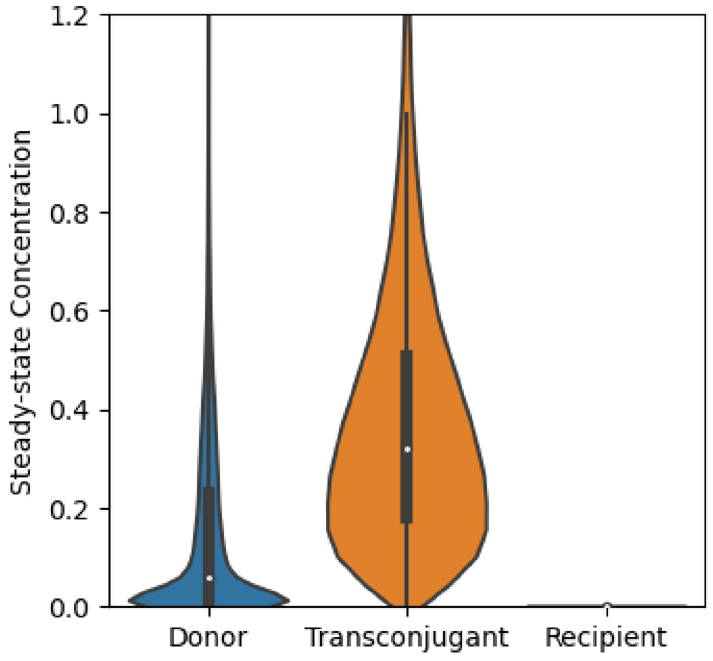
[5]:
```
Donor = SA_spec.results[:,0]
Transconjugant = SA_spec.results[:,1]
Recipient = SA_spec.results[:,2]
```

```
fig, ax = plt.subplots(1,1, figsize=(4,4))

width = .4

ax.bar(0, np.mean(Donor), width, edgecolor="black", yerr = np.std(Donor))
ax.bar(1, np.mean(Transconjugant), width, edgecolor="black", yerr = np.
 ↪std(Transconjugant))
ax.bar(2, np.mean(Recipient), width, edgecolor="black", yerr = np.
 ↪std(Recipient))

ax.set_ylabel('Steady State Abundance')
ax.set_xticks(range(3), SA_spec['outputs'])

plt.savefig(f'tmp_figs/output_barchart_conjugation_{conjugation}.svg')
```



[6]:
```
fig, ax = plt.subplots(1,1, figsize=(4,4))

sns.violinplot(data=SA_spec.results, scale='count')

ax.set_xticks(range(3), SA_spec['outputs'])
ax.set_xlim([-.5,2.5])

ax.set_ylabel('Steady-state Concentration')
ax.set_ylim([0,1.2])
```

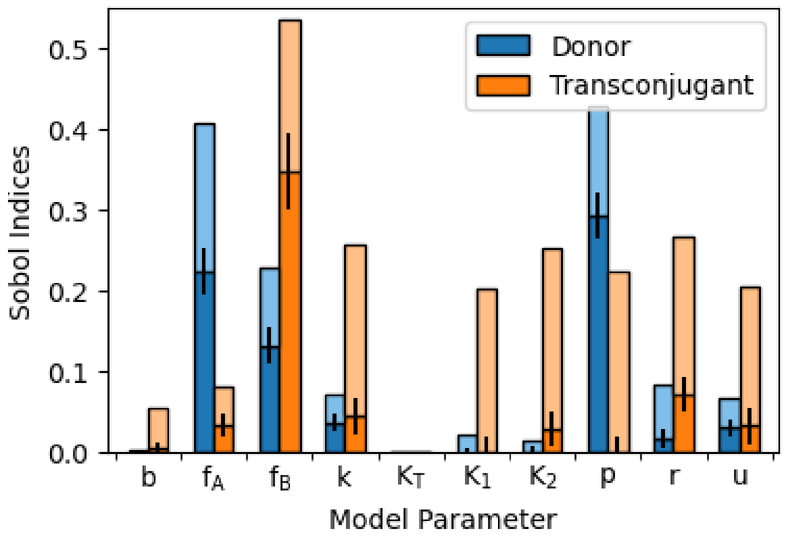
```
plt.savefig(f'tmp_figs/violin_conjugation_{conjugation}.svg')
```



A natural question is which parameters contribute to this variance. We'll let SALib do the work on this by providing various **Sobol indices**. For a given parameter, the first-order Sobol index `S1` indicates the variance in output individually attributable to that parameter. Variance in the output may also arise from interactions between parameters. This **higher-order** effect is be captured in the total-order Sobol index `ST`.

[7]:
```
SA_spec.analyze_sobol(print_to_console=False, calc_second_order=True)
indices = SA_spec.analysis

donor_first = indices['Donor']['S1']
trans_first = indices['Transconjugant']['S1']
recip_first = indices['Recipient']['S1']

donor_higher = indices['Donor']['ST'] - indices['Donor']['S1']
trans_higher = indices['Transconjugant']['ST'] - indices['Transconjugant']['S1']
recip_higher = indices['Recipient']['ST'] - indices['Recipient']['S1']

fig, ax = plt.subplots(1,1, figsize=(4.5,3))

width = .3

xlabels = ['$b$', '$f_A$', '$f_B$', '$k$', '$K_T$', '$K_1$', '$K_2$', '$p$',␣
 ↪'$r$', '$u$']
```

```
x = np.arange(len(xlabels))

if conjugation:
    ax.bar(x-.5*width, donor_first, width, label='Donor', yerr =␣
 ↪indices['Donor']['S1_conf'], edgecolor="black")
    ax.bar(x+.5*width, trans_first, width, label='Transconjugant', yerr =␣
 ↪indices['Transconjugant']['S1_conf'], edgecolor="black")
#     ax.bar(x+width, recip_first, width, label='Recipient', yerr =␣
 ↪indices['Recipient']['S1_conf'], edgecolor="black")

    ax.bar(x-.5*width, donor_higher, width, bottom=donor_first,␣
 ↪color='#7fbee9', edgecolor="black")
    ax.bar(x+.5*width, trans_higher, width, bottom=trans_first,␣
 ↪color='#ffbf86', edgecolor="black")
#     ax.bar(x+width, recip_higher, width, bottom= recip_first,␣
 ↪color='#87de87', edgecolor="black")

else:
    ax.bar(x, np.append(0,recip_first), width, color='#2ca02c',␣
 ↪label='Recipient', yerr = np.append(0,indices['Recipient']['S1_conf']),␣
 ↪edgecolor="black")
    ax.bar(x, np.append(0,recip_higher), width, bottom=np.append(0,␣
 ↪recip_first), color='#87de87', edgecolor="black")


ax.tick_params(axis='x', which='major', length=0)
ax.tick_params(which='minor', length=3)
ax.set_xticks(x, xlabels, minor=False)
ax.xaxis.set_minor_locator(AutoMinorLocator(2))

ax.set_xlim([-2*width, 9+2*width])

ax.set_xlabel('Model Parameter')
ax.set_ylabel('Sobol Indices')
ax.set_ylim([0,0.55])

ax.legend(loc='best')

plt.savefig("tmp_figs/param_sensitivity.svg")
```



```
[8]: donor_second = np.matrix(indices['Donor']['S2'])
     trans_second = np.matrix(indices['Transconjugant']['S2'])


     # print(donor_second.round(decimals=3))
     print(np.nansum(donor_second,axis=1))

     # print(trans_second.round(decimals=3))
     print(np.nansum(trans_second,axis=1))
```

```
[[-1.37833269e-02]
 [ 2.64036634e-01]
 [ 9.80427702e-02]
 [-6.95799192e-02]
 [ 4.75766501e-05]
 [ 5.91418663e-03]
 [ 7.39066894e-03]
 [ 6.62424145e-03]
 [ 1.33187716e-02]
 [ 0.00000000e+00]]
[[-4.49904504e-02]
 [ 4.23669688e-02]
 [-1.24904800e-01]
 [-1.06609778e-01]
 [-2.99898560e-05]
 [ 1.31900511e-02]
 [ 3.95648360e-02]
 [ 3.54464816e-02]
 [ 9.48516682e-03]
[ 0.00000000e+00]]
```