Synthesizing Stochasticity in Biochemical Systems

Brian Fett
University of Minnesota
200 Union St. S.E.
Minneapolis, MN 55455
fett@umn.edu

Jehoshua Bruck
California Institute of Technology
Mail Code 136-93
Pasadena, CA 91125
bruck@paradise.caltech.edu

Marc D. Riedel
University of Minnesota
200 Union St. S.E.
Minneapolis, MN 55455
mriedel@umn.edu

ABSTRACT

Randomness is inherent to biochemistry: at each instant, the sequence of reactions that fires is a matter of chance. Some biological systems exploit such randomness, choosing between different outcomes stochastically – in effect, hedging their bets with a portfolio of responses for different environmental conditions. In this paper, we discuss techniques for synthesizing such stochastic behavior in engineered biochemical systems. We propose a general method for designing a set of biochemical reactions that produces different combinations of molecular types according to a specified probability distribution. The response is precise and robust to perturbations. Furthermore, it is programmable: the probability distribution is a function of the quantities of input types. The method is modular and extensible. We discuss strategies for implementing various functional dependencies: linear, logarithmic, exponential, etc. This work has potential applications in domains such as biochemical sensing, drug production, and disease treatment. Moreover, it provides a framework for analyzing and characterizing the stochastic dynamics in natural biochemical systems such as the lysis/lysogeny switch of the *lambda* bacteriophage.

1. INTRODUCTION

Increasingly, biology is becoming a *computational* science, as modeling and simulation are applied alongside experimental work in the lab [1]. Furthermore, with the advent of techniques for synthesizing and manipulating genetic material, it is striving to become an *engineering* discipline. In the nascent field of *synthetic biology*, researchers aim to create entirely new biological functions by modifying and integrating biological components in a systematic way [2]. The potential impacts are far-reaching. Recent feats of synthetic biology include cellulosic ethanol [3], anti-malarial drugs [4], and tumor detection [5].

By custom-designing the genetic material of organisms such as yeast and *E. coli*, it is possible to directly synthesize biochemistry for applications. In principle, the approach could produce biochemical reactions of nearly any form. However, designing a set of reactions to implement a desired functionality – efficiently and robustly – is a challenging problem.

Biochemical systems are typically characterized through computationally intensive Monte Carlo simulations [6, 7]. Cook and Bruck have studied biochemical reactions from a theoretical perspective, for instance proving universality [8]. Samoilov et al. have discussed the implementation of signal processing functions [9].

In this paper, we focus on *stochasticity* in biochemical systems, tackling the synthesis of probabilistic behavior as a general design problem. We propose a method for designing a set of biochemical reactions that produces different combinations of molecular types according to a specified probability distribution. The response is precise and robust to perturbations. Furthermore, it is programmable: the probability distribution is a function of the quantities of input types. The method is modular and extensible. We discuss strategies for implementing various functional dependencies: linear, logarithmic, exponential, etc. More complex functions can be built by combining the modules for these basic functions.

1.1 Randomness in Biochemistry

Interesting cellular chemistry typically involves complex molecules such as proteins and enzymes. Within the confines of a cell, the *quantities* of such molecules are often surprisingly small: on the order of tens, hundreds, or thousands of molecules of each type. At this scale, individual reactions matter, and the problem must be analyzed discretely [6]. The complexity stems from the dynamics at play among the multitude of coupled reactions. Randomness is inherent: at

each instant, the exact sequence of reactions that fires next is a matter of chance.

Indeed, a biochemical system behaves as a discrete, probabilistic, finite-state machine – or *Markov* chain. The *state* consists of the molecular quantities measured in *whole* (i.e., non-negative integer) amounts. For instance, with molecular types a, b, and c, the state of the system might be 15 molecules of a, 25 of b, and 0 of c:

$$S_1 = [15, 25, 0].$$

Transitions occur as discrete events when reactions fire. Consider the reaction

$$a + b \xrightarrow{10} 2c.$$

When this reaction fires, one molecule of a is consumed, one of b is consumed, and two of c are produced. (Accordingly, a and b are called the *reactants* and c the *product*.) The new state is

$$S_2 = [14, 24, 2].$$

Each reaction has an associated *rate* (listed above the arrow in our notation). Given several reactions, the probability of each firing is proportional both to its rate and to the quantities of its reactants present. We assume that the rate is constant. (In general, it might be dependent on temperature, cell size, and other physical parameters.)

Certain biochemical systems appear to exploit randomness, choosing between different outcomes with a probability distribution – in effect, hedging their bets with a portfolio of responses. Examples include the *pap pili* epigenetic response of bacteria [10], the lentiviral positive-feedback loop in the HIV virus [11], and the lysis/lysogeny switch of the *lambda* bacteriophage [12]. We discuss the *lambda* switch in detail in Section 3.

1.2 Engineering Stochasticity

As in natural systems, randomness can play a pivotal role in synthetically engineered systems. Consider the following design problem. (While entirely hypothetical, it is in the vein of exciting recent research [5].)

Suppose that bacteria are engineered to invade tumors and produce a drug to kill the cancer cells. The bacteria are engineered to produce the drug in response to a compound that is injected into the cancerous tissue. Until this compound is supplied, the bacteria remain inert. In response to it, each bacterium produces a fixed quantity of the drug. The dosage must be calibrated. If every bacterium responds, then the patient receives too high of a dose of the drug. Instead the correct dosage is achieved if only some fraction of the bacteria respond – say, if m out of the population of n respond.

Yet, all the bacteria are identical and subject to the same environment. How can this be achieved? We could aim to engineer a *probabilistic response* whereby each bacterium produces the drug with probability $\frac{m}{n}$.

Further, suppose that the dosage must have a specific functional dependence on the input quantity of the compound so that it can be adjusted as the treatment progresses. Given an input quantity X, the required probabilistic response might be of the form $p = f(X)$ where $f(X)$ is, say, a logarithmic function.

Figure 1 illustrates the general framework that we adopt for synthesis. The input consists of quantities of certain molecular types. The output consists of the probability distribution of certain discrete outcomes. The design problem is to select biochemical reactions that perform this computation. We use the convention that a lower-case letter, such as x, denotes a molecular *type*; the corresponding upper-case letter, X, denotes the *quantity* of this type.

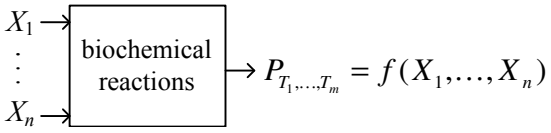

Figure 1: Computational Framework. The input consists of quantities of molecular types $X_1, \ldots, X_n$. The output consists of a probability distribution on discrete outcomes $T_1, \ldots, T_m$ that is some function f of the inputs. The computation is performed by biochemical reactions firing discretely and probabilistically.

In this paper, we tackle the design problem in the abstract – working not with specific molecular types but rather with arbitrary types (a, b, c, etc.). Appealing to the modern toolkit of synthetic biology, we assume that biochemical reactions in suitable types can be engineered [2, 13].

2. MODULAR SYNTHESIS SCHEME

We decompose synthesis into two modules, as shown in Figure 2. First, a deterministic module produces output quantities that are a fixed function of input quantities. Then a stochastic module produces the desired probabilistic response as a function of these outputs. The deterministic module is composed of smaller modules implementing specific functional dependencies. We discuss the stochastic module first.

2.1 Stochastic Module

This module produces the desired probabilistic response as a function of quantities of input types. Consider the following example.

Example 1: Suppose that we have a system with molecular types d_1, d_2, and d_3. We wish to to program the production of these types with the probability distribution

$$p_1 = 0.3, \qquad p_2 = 0.4, \qquad p_3 = 0.3,$$

respectively. To do so, we set up *initializing* reactions:

$$e_1 \xrightarrow{1} d_1, \quad e_2 \xrightarrow{1} d_2, \quad e_3 \xrightarrow{1} d_3.$$

We initialize the system with quantities of e_1, e_2, and e_3 in the desired ratio of $3 : 4 : 3$,

$$E_1 = 30, \qquad E_2 = 40, \qquad E_3 = 30.$$

(Should we want a different probability distribution, we simply change the ratio of these initial quantities.) Given these ratios, the reaction producing d_1 fires first with probability 0.3, the one producing d_2 fires first with probability 0.4, and the one producing d_3 fires first with probability 0.3. We want to cement this initial choice. Accordingly, we set up *reinforcing* reactions:

$$e_1 + d_1 \xrightarrow{10^3} 2d_1, \quad e_2 + d_2 \xrightarrow{10^3} 2d_2, \quad e_3 + d_3 \xrightarrow{10^3} 2d_3.$$

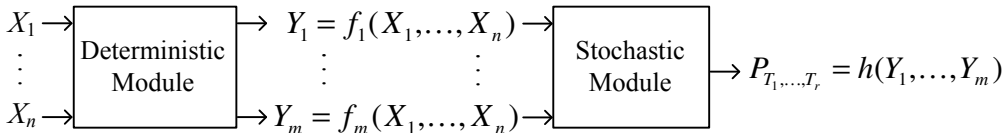

Figure 2: Modular Synthesis. The inputs to the deterministic module are quantities of n types $X_1, \ldots, X_n$. The outputs are quantities of m types $Y_1, \ldots, Y_m$. These quantities are functions $f_1, \ldots, f_m$ of the input quantities. In turn, these outputs are inputs to the stochastic module, which produces a probability distribution on distinct outcomes $T_1, \ldots, T_m$. The distribution is a function g of its input quantities.

Also, we set up *stabilizing* reactions:

$$d_1 + e_2 \xrightarrow{10^3} d_1, \quad d_1 + e_3 \xrightarrow{10^3} d_1, \quad d_2 + e_1 \xrightarrow{10^3} d_2,$$
$$d_2 + e_3 \xrightarrow{10^3} d_2, \quad d_3 + e_1 \xrightarrow{10^3} d_3, \quad d_3 + e_2 \xrightarrow{10^3} d_3.$$

Note that the reinforcing and stabilizing reactions have much higher rates than the initializing reactions. Finally, we set up *purifying* reactions:

$$d_1 + d_2 \xrightarrow{10^6} \varnothing, \quad d_1 + d_3 \xrightarrow{10^6} \varnothing, \quad d_2 + d_3 \xrightarrow{10^6} \varnothing.$$

($\varnothing$ indicates that there are no products that we care about for this reaction.) Note that the purifying reactions have still higher rates.

The reinforcing, stabilizing and purifying reactions ensure that as soon as an initializing reaction fires, producing a molecule of d_i, this choice quickly wins out: the production of more molecules of d_i is encouraged, while the production of the other types d_j, $j \neq i$, is strongly inhibited. So the firing probabilities for the initializing reactions at the outset dictate the probability distribution of the final outcome.

2.1.1 The Set of Reactions

The stochastic module consists of five categories of reactions. The rates are assumed to be similar for all the reactions in each category; however, the rates between categories must be different: some categories are slow and other are comparatively fast, as explained in Section 2.1.3.

For all of the categories, the subscripts i and j run over the number of desired outcomes. For each outcome, we have an *input* type e; a *catalyst* type d; *food* types f; and *output* types o.

Initializing Reactions

$$\forall\, i: \qquad e_i \xrightarrow{k_i} d_i$$

These reactions initiate the response with the production of a catalyst type. They are the slowest reactions in the system. The first one to fire generally determines the outcome. (As discussed in Section 2.1.3, the likelihood of a different outcome is vanishingly small.)

Reinforcing Reactions

$$\forall\, i: \qquad d_i + e_i \xrightarrow{k'_i} 2d_i$$

These reactions amplify the choice made by the initializing reactions, increasing the quantity of the catalyst type. (The quantity of catalyst that is produced here is limited by amount of the input type that is supplied. It could be limited some other way, but this is convenient.)

Stabilizing Reactions

$$\forall\, j \neq i: \qquad d_i + e_j \xrightarrow{k''_{ij}} d_i$$

These reactions consume all input types other than the one that was selected. So they inhibit competing outcomes.

Purifying Reactions

$$\forall\, j \neq i: \qquad d_i + d_j \xrightarrow{k'''_{ij}} \varnothing$$

These reactions quickly suppress any competing catalyst types. They are the fastest reactions in the system. If ever there are multiple catalyst types present, those in the minority are quickly wiped out, whereas those in the majority are only slightly weakened in number.

Working Reactions

$$\forall i, \ell_i: \qquad d_i + f_{\ell_i} \xrightarrow{k''''_i} d_i + o_{\ell_i}$$

These reactions take the decision made by the initializing reactions and turn it into action: they produce output types in the desired quantity. Several output types in differing proportions can be created for each catalyst type. This can be accomplished with different working reactions operating on the same catalyst type. Alternatively, a single working reaction can be set up with multiple output types in the desired proportions.

2.1.2 Initial Quantities

The probability of the i-th initializing reaction firing first is proportional to its rate, k_i, and to the quantity E_i of its input type e_i. Accordingly, we can *program* the firing probabilities by setting the ratio of the initial quantities:

$$\forall\, i: \qquad p_i = \frac{E_i k_i}{\sum_{\forall j} E_j k_j}.$$

Thus, the initial quantities of the input types directly determine the probability distribution of the outcomes. At the outset, there are no catalyst types or output types. The initial quantities of the food types are set to the maximum quantity desired for the corresponding output types.

2.1.3 Reaction Rates

The rates should be selected so that the initializing and working reactions are the slowest; the reinforcing and stabilizing reactions comparatively much faster; and the purifying reactions fastest of all:

$$k_i \approx k''''_i \quad \ll \quad k'_i \approx k''_{ij} \quad \ll \quad k'''_{ij}.$$

In order to quantify the effect of this separation in the rates, let us choose a multiplicative factor, γ, and set the rates as follows:

$$\gamma k_i = k_i' = k_{ij}'' = k_{ij}'''/\gamma = \gamma k_i''''. \quad (1)$$

Define an *error* to be the case where the first initializing reaction to fire does *not* determine the final outcome; instead, a different catalyst type wins out.

We characterize this error as a function of γ. More specifically, we set up the reactions described in Section 2.1.1 for $i = 1, 2, 3$, with each $k_i = 1$ and each $k_i', k_{ij}'', k_{ij}'''$, and k_i'''' set according to Equation 1. We set the initial quantity of each input type to 100. We assume that a working reaction needs to fire 10 times for us to declare an outcome.

We performed Monte Carlo simulations and obtained the results shown in Figure 3. The graph shows that the error can be made vanishingly small by increasing the separation in the rates.

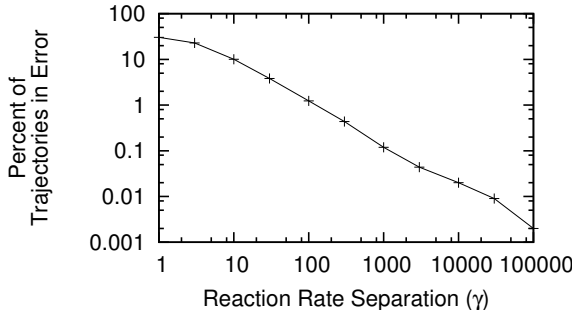

Figure 3: Error Analysis for the Stochastic Module. Monte Carlo simulations with 100,000 trials were performed for different values of γ. The graph gives the percentage of trials that resulted in error.

2.2 Deterministic Module

The *deterministic* module provides flexibility in programming the probabilistic response. It consists of a series of submodules implementing specific functional dependencies.

Example 2: In Example 1, a stochastic module implemented the mutually exclusive production of types d_1, d_2, and d_3. However, suppose we need a probabilistic response with a specific *functional* dependence on the quantity of input types x_1 and x_2:

$$\begin{aligned} p_1 &= 0.3 + 0.02X_1 - 0.03X_2 \\ p_2 &= 0.4 + 0.03X_2 \\ p_3 &= 0.3 - 0.02X_1. \end{aligned}$$

To achieve this, we add some "preprocessing". We set up reactions that modify the probabilities of the initializing reactions occurring:

$$2e_3 + x_1 \xrightarrow{10^3} 2e_1, \qquad 3e_1 + x_2 \xrightarrow{10^3} 3e_2.$$

These two reactions enforce the exact dependence that we need on the quantities X_1 and X_2.

2.2.1 Functional Modules

In describing the functions that the modules implement, we add subscripts to the quantities of molecular types to denote *when* these quantities exist: zero indicates that this is the initial quantity, whereas infinity indicates that it is the quantity after the module has finished.

Linear

$$\boxed{\alpha Y_\infty = \beta X_0}$$

This module produces a quantity of an output type that is proportional to the quantity of an input type. For integer coefficients α and β, the reaction is:

$$\alpha x \rightarrow \beta y.$$

Exponentiation:

$$\boxed{Y_\infty = 2^{X_0}}$$

This module consumes molecules of an input type one at a time, doubling the quantity of an output type for each. Its behavior is described by the following pseudocode:

```
1  ForEach x {
2    Y = 2 * Y;
3  }.
```

The reactions are:

$$\begin{aligned} x &\xrightarrow{\text{slow}} a \\ a + y &\xrightarrow{\text{faster}} a + 2y' \\ a &\xrightarrow{\text{fast}} \varnothing \\ y' &\xrightarrow{\text{medium}} y. \end{aligned}$$

Initially, Y is one and all other quantities (except X) are zero.

Logarithm:

$$\boxed{Y_\infty = log_2(X_0)}$$

This module is similar to the exponentiation module, except that instead of doubling the output, the input is forced to halve itself; each time it does so, the output is incremented by one. Its behavior is described by the following pseudocode:

```
1  While Not(X==1) {
2    X = X/2;
3    Y = Y+1;
4  }.
```

The reactions are:

$$\begin{aligned} b &\xrightarrow{\text{slow}} a + b \\ a + 2x &\xrightarrow{\text{faster}} c + x' + a \\ 2c &\xrightarrow{\text{faster}} c \\ a &\xrightarrow{\text{fast}} \varnothing \\ x' &\xrightarrow{\text{medium}} x \\ c &\xrightarrow{\text{medium}} y. \end{aligned}$$

Initially, B is a small but non-zero quantity and all other quantities (except X) are zero.

Raising to a Power:

$$\boxed{Y_\infty = X_0^{P_0}}$$

This module implements the raising of an input to a power based on the computations $X^P = \prod_P(X)$ and $\alpha X = \sum_X(\alpha)$. This implies a double loop:

```
1  ForEach p {
2    ForEach x {
3      D = D + Y;
4    }
5    Y = D; D = 0;
6  },
```

The reactions are:

$$p \xrightarrow{\text{slowest}} a \tag{2}$$

$$a + x \xrightarrow{\text{medium}} b + a + x' \tag{3}$$

$$b + y \xrightarrow{\text{fastest}} y' + d + b \tag{4}$$

$$b \xrightarrow{\text{faster}} \varnothing \tag{5}$$

$$y' \xrightarrow{\text{fast}} y \tag{6}$$

$$a \xrightarrow{\text{slow}} e \tag{7}$$

$$e + y \xrightarrow{\text{faster}} e \tag{8}$$

$$e + x' \xrightarrow{\text{faster}} e + x \tag{9}$$

$$e \xrightarrow{\text{fast}} \varnothing \tag{10}$$

$$d \xrightarrow{\text{slower}} y. \tag{11}$$

Initially, Y is one and all other quantities (excluding X and P) are zero. Line 1 corresponds to Reaction 2 and line 2 to Reaction 3; these introduce "loop" types a and b, respectively, that cause fast reactions to occur until depletion. These types then degrade (Reactions 5, 7, and 10). Slow reactions allow the state to reset for another iteration. Line 3 is performed by Reaction 4. Reaction 6 resets the state for the next iteration of the inner loop. Reaction 8 reinitializes the quantity of y to zero. Line 5 is performed by Reaction 11. Reaction 9 resets x for the next iteration of the outer loop.

Isolation:

$$\boxed{Y_\infty = 1}$$

This module is used to enforce an initial state consisting of a single molecule of some type. It is needed as a precursor for exponentiation and raising to a power. The reactions are:

$$c + 2y \xrightarrow{\text{fast}} c + y \tag{12}$$

$$c \xrightarrow{\text{slow}} \varnothing. \tag{13}$$

The module requires only that the quantities of types y and c be non-zero at the outset. Upon completion, there is exactly one molecule of type y and none of type c. Note that the molecules of c are all consumed, so the molecules of y can serve as inputs to other modules, provided that Reaction 13 completes in time.

2.2.2 *Combining Modules*

A simple example of combining linear and logarithmic modules is given in Section 3.2. With the linear and raising-to-a-power modules, our scheme can be used to implement arbitrary polynomial functions; hence, in principle, it could be used to approximate complex functions through Taylor series expansions.

Note that in our definitions above, the molecular types are specific to each module (e.g., each x appearing in a different module should be considered a distinct type when combining these). Also, the rates – "fast" vs. "slow", etc. – are relative *within* the modules. When combining modules, one might have to choose reactions with appropriate separations in their rates. (In some cases, the slowest reaction in one module might be *faster* than the fastest reaction in the next.)

3. APPLICATION: MODELING THE LAMBDA BACTERIOPHAGE

We apply our synthesis method to fit the data from a well-known biological model, that of the lysis/lysogeny switch of the *lambda* bacteriophage [12]. Our goal is to demonstrate that our method can accurately capture the stochastic behavior observed in natural biological systems. This is analogous to the concept of"reduced-order modeling" in engineering analysis: the input/output behavior is maintained while the internal dynamics are lost. We do not present any biological interpretation of the results. However, we point out that our synthetic model is more compact and perhaps more robust to specific perturbations than the natural model.

3.1 Natural Model

The *lambda* bacteriophage is a virus that infects the *E. coli* bacteria. It chooses one of two survival strategies: either it integrates its genetic material with that of its host and then replicates when the bacterium divides (termed *lysogeny*); or it manipulates the molecular machinery of its host to make many copies of itself, killing the bacterium in the process, and thereby releasing its progeny into the environment (termed *lysis*).

The biological model for this behavior consists of an elaborate set of 117 reactions in 61 molecular types [12]. We focus on a single input type, *moi*, that plays a crucial role in the viral response. (It correlates with the number of copies of the virus that have infected the cell.) We analyze the probabilistic response for the production of two output types, *cro2* and *ci2*. The production of the former corresponds to lysis; the production of the latter to lysogeny. (The outcomes are judged according to threshold values: 55 for *cro2* and 145 for *ci2*.)

We characterized the probabilistic response of the model with Monte Carlo simulations. Sweeping the input type *moi* across a range of values, we recorded the percentage of trials that resulted in each outcome. From this data, we performed a curve fit. The result:

$$P(\mathit{lysis}) = 15 + 6\log_2(MOI) + \frac{MOI}{6}. \tag{14}$$

The data points and the curve fit are shown in Figure 5. We assume that one or the other outcome always occurs, so $P(\mathit{lysogeny}) = 1 - P(\mathit{lysis})$.

3.2 Synthetic Model

Applying our synthesis methodology to fit Equation 14, we obtain a model with 19 reactions in 17 types, given in Figure 4. Deterministic modules produce the linear and logarithmic dependence. A stochastic module, with its five

(fan-out)	moi	$\xrightarrow{10^9}$	$x_1 + x_2$
(linear)	$6x_2$	$\xrightarrow{10^9}$	y_1
(logarithm)	b	$\xrightarrow{10^{-3}}$	$b + a$
	$a + 2x_1$	$\xrightarrow{10^6}$	$a + x_1' + c$
	$2c$	$\xrightarrow{10^6}$	c
	a	$\xrightarrow{10^3}$	$\varnothing$
	x_1'	$\xrightarrow{1}$	x_1
(linear)	c	$\xrightarrow{1}$	$6y_2$
(assimilation)	$e_1 + y_2$	$\xrightarrow{10^9}$	e_2
	$e_2 + y_1$	$\xrightarrow{10^9}$	e_1
(initializing)	e_1	$\xrightarrow{10^{-9}}$	d_1
	e_2	$\xrightarrow{10^{-9}}$	d_2
(reinforcing)	$e_1 + d_1$	$\xrightarrow{1}$	d_1
	$e_2 + d_2$	$\xrightarrow{1}$	d_2
(stabilizing)	$e_2 + d_1$	$\xrightarrow{1}$	d_1
	$e_1 + d_2$	$\xrightarrow{1}$	d_2
(purifying)	$d_1 + d_2$	$\xrightarrow{10^9}$	$\varnothing$
(working)	$d_1 + f_1$	$\xrightarrow{10^{-9}}$	$d_1 + cro2$
	$d_2 + f_2$	$\xrightarrow{10^{-9}}$	$d_2 + ci2$

Figure 4: Synthetic Model. These reactions implement a probabilistic response that fits that of the lysis/lysogeny decision in the *lambda* bacteriophage.

categories of reactions, produces the probabilistic response. Some simple additional reactions are used to glue the modules together (these are labeled *fan-out* and *assimilation*).

As with the natural model, the input type is *moi* and the output types are *cro2* and *ci2*. The initial quantities of e_1 and e_2 are 15 and 85, respectively (chosen to fit the constant of 15 in Equation 14). The initial quantities of f_1 and f_2 are set sufficiently high to ensure that the appropriate working reactions bring the output molecules above their thresholds of 55 and 145, respectively. The initial quantity of b is set to one, and all other quantities are set to zero.

We characterized the probabilistic response of the synthetic model with Monte Carlo simulations. The results are shown in Figure 5. As can be seen, it implements a close fit to the natural model.

4. DISCUSSION

We are exploring the implementation of our synthesis methodology with parts from the MIT BioBricks repository [13]. Looking forward, our work could play an important role in domains such as biochemical sensing, drug production, and disease treatment. However, the impetus for this research is more immediate: we are interested in analyzing and characterizing the stochastic dynamics of natural biological systems in a rigorous and systematic way – applying engineering concepts such as abstraction and reduced-order modeling. In addition to the *lambda* switch discussed here, we are studying a number of other biological models, including the lentiviral positive-feedback loop in the HIV virus [11] and the pheromone-response pathway in yeast [14].

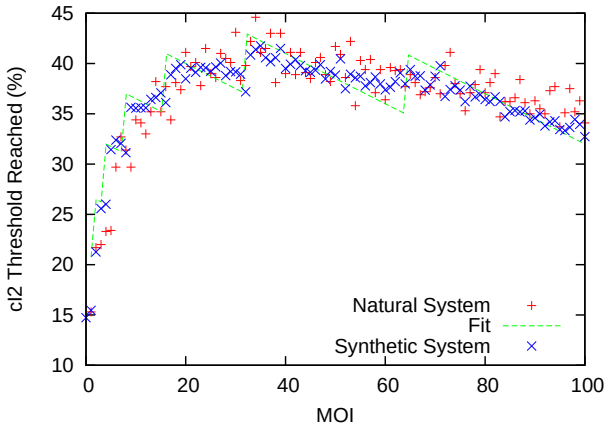

Figure 5: Probabilistic Response. The figure shows the result of Monte Carlo simulations for both for the natural model and for our synthetic model, sweeping the quantity of the input type *moi* from 1 through 10. It shows curve fits for both sets of data.

5. REFERENCES

[1] D. Endy and R. Brent,"Modelling Cellular Behaviour," *Nature*, Vol. 409, pp. 391–395, 2001.

[2] D. Endy, "Foundations For Engineering Biology," *Nature*, Vol. 438, pp. 449–453, 2005.

[3] M. Sedlak and N. Ho, "Production of Ethanol from Cellulosic Biomass Hydrolysate Using Genetically Engineered Yeast," *Applied Biochemistry & Biotechnology,* Vol. 114, No. 1-3, pp. 403–416, 2004.

[4] D.-K. Ro et al., "Production of the Antimalarial Drug Precursor Artemisinic Acid in Engineered Yeast," *Nature*, Vol. 440, pp. 940–943, 2006.

[5] J. Anderson, E. Clarke, A. Arkin, and C. Voigt, "Environmentally Controlled Invasion of Cancer Cells by Engineered Bacteria," *Journal of Molecular Biology*, Vol. 355, No. 4, pp. 619–627, 2006.

[6] D. Gillespie, "Exact Stochastic Simulation of Coupled Chemical Reactions," *Journal of Physical Chemistry*, Vol. 81, No. 25, pp. 2340–2361, 1977.

[7] M. Gibson and J. Bruck, "Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels," *Journal of Physical Chemistry A*, No. 104, pp. 1876–1889, 2000.

[8] M. Cook, "Networks of Relations," *Ph.D. Dissertation*, Advisor J. Bruck, Caltech, 2005.

[9] M. Samoilov, A. Arkin, and J. Ross, "Signal Processing by Simple Chemical Systems,"Journal of Physical Chemistry, Vol. 106, pp. 10205–10221, 2002.

[10] A. Hernday, B. Braaten, and D. Low," The Intricate Workings of a Bacterial Epigenetic Switch," *Advances in Experimental Medicine & Biology*, Vol. 547, No. 83-9, 2004.

[11] L. Weinberger, J. Burnett, J. Toettcher, A. Arkin, and D. Schaffer, "Stochastic Gene Expression in a Lentiviral Positive-Feedback Loop: HIV-1 Tat Fluctuations Drive Phenotypic Diversity," *Cell*, Vol. 122, pp. 169-182, 2005.

[12] A. Arkin, J. Ross, and H. McAdams, "Stochastic Kinetic Analysis of Developmental Pathway Bifurcation in Phage λ-Infected E. Coli Cells," *Genetics*, Vol. 149, No. 1633, 1998.

[13] BioBricks Parts List, *MIT Registry of Standard Biological Parts*,`http://parts.mit.edu`.

[14] I. Herskowitz, "Life Cycle of the Budding Yeast *Saccharomyces cerevisiae*," *Microbiological Reviews*, Vol. 52, No. 4, pp. 536-553, 1988.