Systems Biology Assignment

Riccardo Conci (rc667), Georgeos Hardo (gh464), Trevor Manz (tjm74), Laura Martens (ldm41)

July 24, 2019

Question I

In this assignment, we consider the following mRNA dynamics of two genes that form an operon which is regulated by a transcription factor x_1. In this example x_2 and x_3 denote the mRNA species and τ_2 and τ_3 the expected lifetimes, respectively.

$$\begin{array}{lll} \text{TF translation: } x_1 \xrightarrow{\alpha} x_1+1 & \text{mRNA transcription: } x_2 \xrightarrow{\lambda x_1} x_2+1 & \text{mRNA transcription: } x_3 \xrightarrow{\lambda x_1} x_3+1 \\ \text{TF proteolysis: } x_1 \xrightarrow{x_1/\tau_1} x_1-1 & \text{mRNA degradation: } x_2 \xrightarrow{x_2/\tau_2} x_2-1 & \text{mRNA degradation: } x_3 \xrightarrow{x_3/\tau_3} x_3-1 \end{array} \tag{1}$$

The normalised (co)variances of the system η_{ij} can be obtained by solving the fluctuation dissipation relation with respect to η:

$$M\eta + (M\eta)^{\mathrm{T}} = D \quad \text{with} \quad \eta = \begin{pmatrix} \eta_{11} & \eta_{12} & \eta_{13} \\ \eta_{12} & \eta_{22} & \eta_{23} \\ \eta_{13} & \eta_{23} & \eta_{33} \end{pmatrix}, \tag{2}$$

where the diffusion matrix D is given by:

$$D_{ij} = \frac{2}{\tau_i}\frac{\langle s_{ij}\rangle}{\langle x_j\rangle} + \frac{2}{\tau_j}\frac{\langle s_{ji}\rangle}{\langle x_i\rangle} \tag{3}$$

and M is given by:

$$M_{ij} = \frac{H_{ij}}{\tau_i} \tag{4}$$

with:

$$H_{ij} = \frac{\partial \ln\left(R_i^-/R_i^+\right)}{\partial \ln x_j} \tag{5}$$

Each event only adds or removes one molecule, so we have $\langle s_1\rangle = \langle s_2\rangle = \langle s_3\rangle = 1$. There is no event that simulatenously removes or adds two different molecules, so $\langle s_{ij}\rangle = 0,\ i \neq j.$:

$$D = \begin{pmatrix} \frac{2}{\langle x_1\rangle\tau_1} & 0 & 0 \\ 0 & \frac{2}{\langle x_2\rangle\tau_2} & 0 \\ 0 & 0 & \frac{2}{\langle x_3\rangle\tau_3} \end{pmatrix}$$

All fluxes are first or zero order, so it follows:

$$H = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \Longrightarrow M = \begin{pmatrix} \frac{1}{\tau_1} & 0 & 0 \\ -\frac{1}{\tau_2} & \frac{1}{\tau_2} & 0 \\ -\frac{1}{\tau_3} & 0 & \frac{1}{\tau_3} \end{pmatrix}$$

Inserting D and M into (2) yields six equations for our six unknown (co)variances:

$$
\begin{array}{ll}
1.\ \eta_{11} = \dfrac{1}{\langle x_1 \rangle} & 2.\ \eta_{22} = \dfrac{1}{\langle x_2 \rangle} + \eta_{12} \\
3.\ \eta_{33} = \dfrac{1}{\langle x_3 \rangle} + \eta_{13} & 4.\ \eta_{11} = \dfrac{\tau_2 + \tau_1}{\tau_1}\eta_{12} \\
5.\ \eta_{11} = \dfrac{\tau_3 + \tau_1}{\tau_1}\eta_{13} & 6.\ (\tau_2 + \tau_3)\eta_{23} = \tau_3\eta_{13} + \tau_2\eta_{12}
\end{array}
$$

Solving for the (co)variances gives:

$$
\begin{aligned}
\eta_{11} &= \frac{\text{var}(x_1)}{\langle x_1 \rangle^2} &&= \frac{1}{\langle x_1 \rangle} \\
\eta_{22} &= \frac{\text{var}(x_2)}{\langle x_2 \rangle^2} &&= \frac{1}{\langle x_2 \rangle} + \frac{1}{\langle x_1 \rangle} \cdot \frac{\tau_1}{(\tau_1 + \tau_2)} \\
\eta_{33} &= \frac{\text{var}(x_3)}{\langle x_3 \rangle^2} &&= \frac{1}{\langle x_3 \rangle} + \frac{1}{\langle x_1 \rangle} \cdot \frac{\tau_1}{(\tau_1 + \tau_3)} \\
\eta_{12} &= \frac{\text{cov}(x_1, x_2)}{\langle x_1 \rangle \langle x_2 \rangle} &&= \frac{1}{\langle x_1 \rangle} \cdot \frac{\tau_1}{(\tau_1 + \tau_2)} \\
\eta_{13} &= \frac{\text{cov}(x_1, x_3)}{\langle x_1 \rangle \langle x_3 \rangle} &&= \frac{1}{\langle x_1 \rangle} \cdot \frac{\tau_1}{(\tau_1 + \tau_3)} \\
\eta_{23} &= \frac{\text{cov}(x_2, x_3)}{\langle x_2 \rangle \langle x_3 \rangle} &&= \frac{1}{\langle x_1 \rangle} \cdot \frac{\tau_1\left(2\tau_2\tau_3 + \tau_1\left(\tau_2 + \tau_3\right)\right)}{\left(\tau_1 + \tau_2\right)\left(\tau_1 + \tau_3\right)\left(\tau_2 + \tau_3\right)}
\end{aligned}
$$

The results are exact, as all the fluxes are first or zero order and therefore are identical to the first order Taylor expansion.

The normalised mRNA variances are composed of an intrinsic noise term and an extrinsic noise term. As both η_{22} and η_{33} are symmetrical and only differ in their respective life times, we will limit our discussion to one term.

$$
\eta_{22} = \underbrace{\frac{1}{\langle x_2 \rangle}}_{\text{intrinsic}} + \underbrace{\frac{1}{\langle x_1 \rangle} \cdot \frac{\tau_1}{(\tau_1 + \tau_2)}}_{\text{extrinsic}}
$$

The noise is dominated by the extrinsic noise in several scenarios. Firstly, if $\tau_1 \gg \tau_2$, meaning that a mRNA is degraded a lot faster than the transcription factor, the ratio of intrinsic versus extrinsic noise is entirely determined by the ratio of $\langle x_1 \rangle$ to $\langle x_2 \rangle$. If we have $\tau_2 \gg \tau_1$, only a small fraction of the transcription factor noise will contribute to the overall noise. If this is, however, coupled with low numbers of the transcription factor, the extrinsic contribution can still be very high. Conversely, the mRNA levels will be dominated by intrinsic low copy-number noise if the mRNA birth rate λ is small compared to the birth rate α of the transcription factor.

Question IIA

The above model is simulated using the Gillespie algorithm (Appendix A.1). The algorithm is implemented as follows:

1. Starting states for all components, $\underline{x}_o$, are set to 1, and time, t, is set to zero.
2. All reaction rates are calculated for system, $r_i(\underline{x})$.
3. The total leaving rate is determined, $r_{\text{TOT}} = \sum_{i=1}^{k} r_i$.
4. A waiting time, t_{RE}, is selected from the exponential distribution of waiting times. This waiting time denotes *when* a reaction occurs for the given iteration.
5. The relative probabilities for each reaction are determined given the total leaving rate, $p_i = \frac{r_i}{r_{\text{TOT}}}$.
6. A reaction, $\underline{\delta}_L$, is sampled given the relative probabilities. The sampled reaction reflects *what* reaction occurs for the given iteration.
7. The system is updated: $\underline{x} \to \underline{x} + \underline{\delta}_L$; $t \to t + t_{\text{RE}}$.
8. Steps 2-7 are repeated for one hundred million iterations, or as many as are necenssary to reach the stationary state.

We repeat the above algorithm for one hundred sets of parameter values, fixing the mRNA lifetimes at $\tau_2 = 2$ and $\tau_3 = 4$ (Appendix A.2). Initially simulations are run for all combinations of $\tau_1 = \{1, 2, 3, 4, 5\}$, $\alpha = \{1, 2, 3, 4\}$, and $\lambda = \{2, 4, 6, 8, 10\}$. However, the combination of parameters did not effectively sample systems ranging from intrinsic to extrinsic noise dominated regimes, yielding only systems with less than 45% intrinsic noise and more than 55% extrinsic noise. Therefore we repeated the simulations for $\tau_1 = \{0.01, 0.1, 1, 5, 50\}$, $\alpha = \{0.1, 1, 5, 10, 100\}$, and $\lambda = \{0.1, 1, 5, 50\}$ and show that the systems sampled span the full range of intrinsic vs extrinsic noise regimes for X_2 and X_3 (Figure 1).

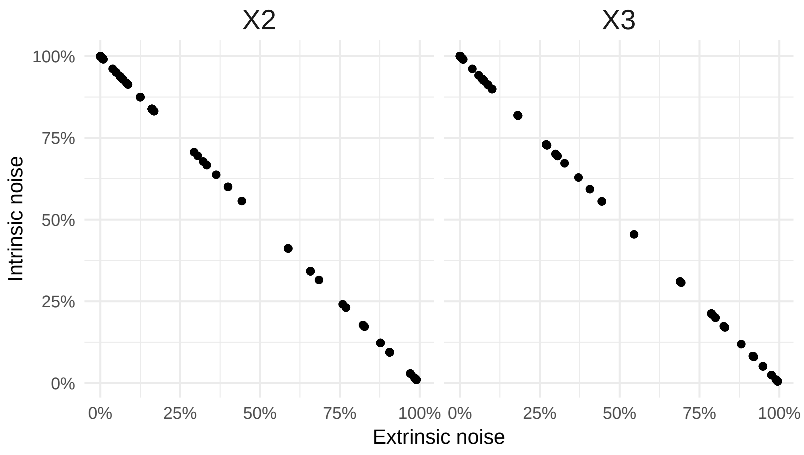


Figure 1: Percent of intrinsic versus the extrinsic contributions to the mRNA fluctuations. Each point reflects a unique combination of $\tau_1 = \{0.01, 0.1, 1, 5, 50\}$, $\alpha = \{0.1, 1, 5, 10, 100\}$, and $\lambda = \{0.1, 1, 5, 50\}$.

It is important to calculate the time weighted (tw) mean of each species over the period of a trace. This is given by:

$$\langle x_i \rangle_{\text{tw}} = \frac{\sum_{k=1}^{n} t_k x_{i,k}}{\sum_{k=1}^{n} t_k} \tag{6}$$

Where x_i is the species in question, n is the number of time-steps, k is the index of the time-step in question, t

is the size of the time-step, and $x_{i,k}$ is the value of species x_i in time-step t_k.

The time weighted variance is therefore given by:

$$\sigma^2_{i,\mathrm{tw}} = \frac{\sum_{k=1}^{n} t_k \left(x_{i,k} - \langle x_i \rangle_{\mathrm{tw}}\right)^2}{\sum_{k=1}^{n} t_k} \tag{7}$$

and the time weighted covariance is given by:

$$\mathrm{cov}(x_i, x_j)_{\mathrm{tw}} = \frac{\sum_{k=1}^{n} t_k \left(x_{i,k} - \langle x_i \rangle_{\mathrm{tw}}\right)\left(x_{j,k} - \langle x_j \rangle_{\mathrm{tw}}\right)}{\sum_{k=1}^{n} t_k} \tag{8}$$

Table 1: Example of time-weighted steady state means of a single Gillespie trace for one hundred million iterations of the algorithm with parameters $\alpha = 10$, $\tau_1 = 2$, $\tau_2 = 3$, $\tau_3 = 4$, $\lambda = 4$.

Component	Analytic mean	Gillespie mean	% error	Relative steady state flux
T factor	20	19.995	0.0243	0.000122
mRNA 1	160	159.91	0.0576	0.000166
mRNA 2	320	319.98	0.00655	0.0000445

Table 2: Example of time-weighted variances and covariances of a single Gillesie trace for one hundred million iterations with the same parameters are those described in Table 1.

Steady state value (normalised)	FDT	Gillespie	% error
var(T factor)	0.05	0.049986	0.02808
var(mRNA 1)	0.03125	0.031323	0.2338
var(mRNA 2)	0.0197917	0.01982	0.1433
cov(T factor, mRNA 1)	0.025	0.02504	0.1703
cov(T factor, mRNA 2)	0.0166...	0.01669	0.1445
cov(mRNA 1, mRNA 2)	0.01944...	0.01951	0.3121

Question IIB

As a simulation approaches steady state, any error in the stationary state fluxes e_f should be close to 0:

$$\frac{\langle R_i^+ \rangle - \langle R_i^- \rangle}{\langle x_i \rangle} = e_f \approx 0 \tag{9}$$

Normalising against $\langle x_i \rangle$ ensures one can compare relative errors, which are detected as deviations from zero. The time weighted means of the fluxes of each component were calculated. The relative stationary state flux of most simulations are centered tightly around 0, indicating that the simulations ran for long enough. The x-axis on the three histograms below exaggerates the apparent deviation from 0 stationary state flux of some simulations, but the deviation is still small in absolute terms, and one can be confident that the simulations did indeed run for an adequate length of time.

The asymmetry in the histogram representing the transcription factor's steady state flux is mirrored in the mRNA histograms. This is because when the relative error in the flux balance is negative for the transcription factor, this indicates that the transcription factor spent slightly more time moving downwards towards the mean. That is to say that it spent slightly more time above the mean. This would in turn increase the transcription rate of the mRNAs, resulting in more positive flux than negative flux. Regardless, in absolute terms the deviation is very small, and it occurs only when the stationary state mean of the transcription factor is low. Possible reasons for this are discussed in question IIC.

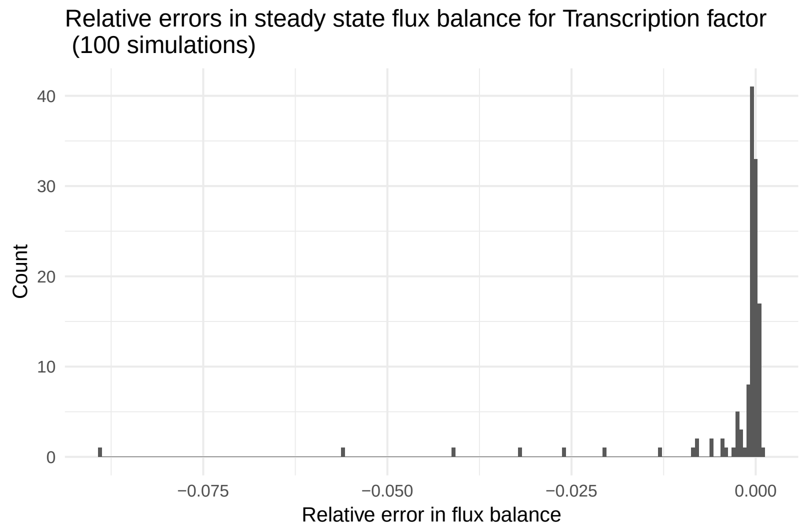
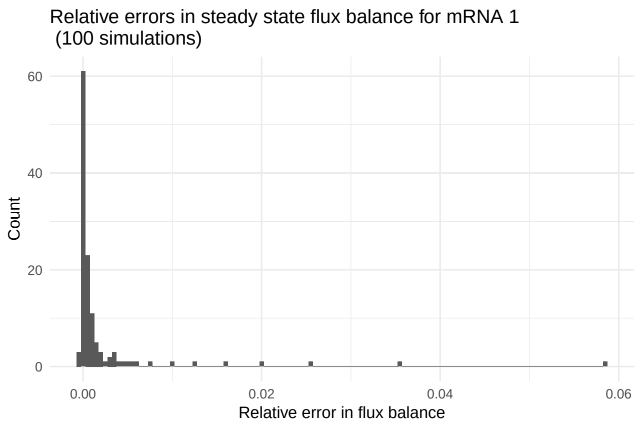
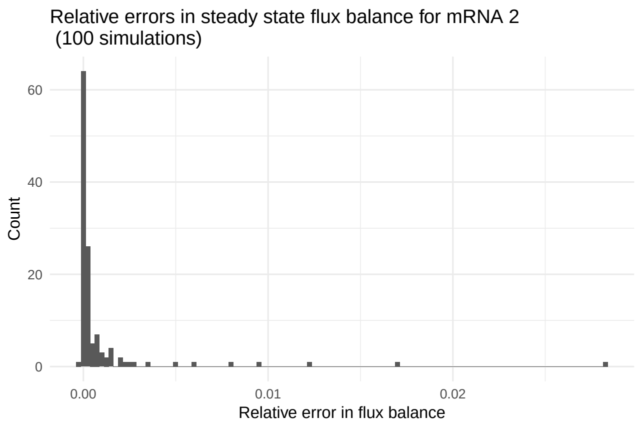


Figure 2: Histogram of relative errors of the flux balance at steady state for the transcription factor.



(a)



(b)

Figure 3: Histogram of relative errors of the flux balance at steady state for the two mRNA species.

Similarly, we checked whether the observed normalised variances and co-variances η_{ij}^{obs} satisfied the analytic expressions derived in Question I. To do this we plotted histogram of the relative deviations e_{ij} from the analytic expression, which should ideally be centered around 0:

$$e_{ij} = \frac{\eta_{ij}^{\text{obs}} - \eta_{ij}}{\eta_{ij}} \tag{10}$$

Overall, the histograms in Figure 4 confirm our expectations, with relative deviations being very close to 0 in both absolute and relative terms. There are again a few values, that show higher deviations, which are probably linked to small values of α and τ_1, the effects of which are explained in more detail in question IIC.

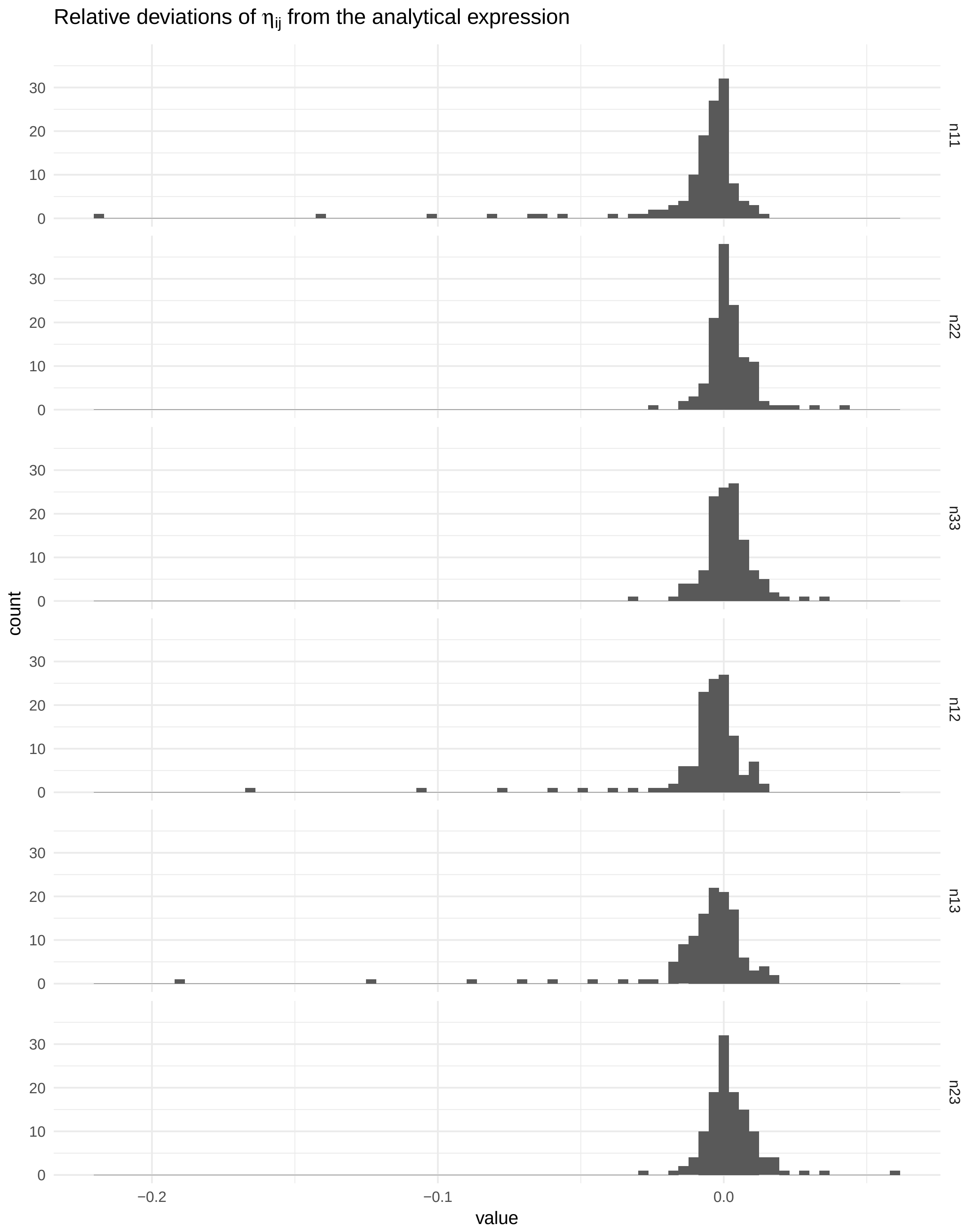


Figure 4: Relative deviations of η_{ij} from the analytical expressions. As expected they are centred around zero with higher deviations for small values for α and τ_1

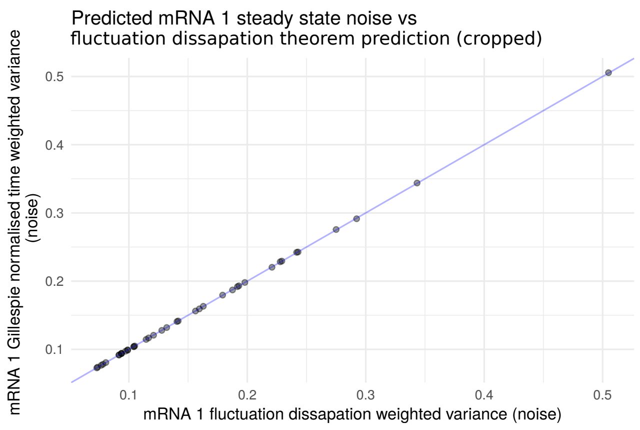
Question IIC

One simulation scenario which needs to be addressed is the case where the transcription factor mean is very low. The mean is very low when the number of transcription factors regularly hits 0. When this occurs, the error between the transcription factor's Gillespie mean and FDT mean become very large. This error in $\langle x_1 \rangle$ then propagates through the calculations for the variances and covariances for all other components, in turn increasing their error too.

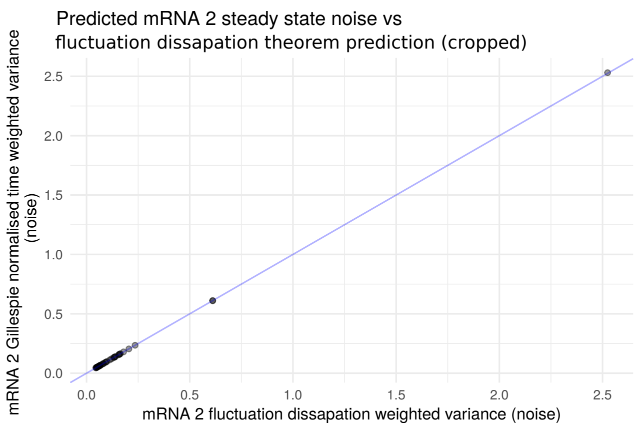
When the transcription factor copy number regularly hits 0 the distribution of states that the simulation is in around the mean predicted fluctuation dissipation theorem becomes asymmetrical. For instance, if the mean copy number of transcription factors as predicted by the FDT is 1, the Gillespie mean is consistently higher, at around 1.059 regardless of trace length (traces tested from 10,000,000 iterations to 1,000,000,000 iterations). This difference gets more pronounced as the transcription factor's FDT mean gets closer to 0. We **suspect** this may be because at a mean of, say, 1, the transcription factor can fluctuate upwards by many copy numbers if the traces are long enough, but can never fluctuate downwards by more than 1 because it hits the floor of 0. Negative transcription factor copy numbers are impossible and so an asymmetry of the distribution of copy numbers around the mean copy number is created by the flooring effect. This asymmetry pushes the Gillespie mean consistently higher.

This is also a likely explanation for the asymmetry in the flux distribution histograms in question IIB. If transcription factors (for a mean of 1) can go arbitrarily high, but can only go as low as 0, negative fluxes which are proportional to the number of transcription factors can also go arbitrarily high, as the transcription factor can exist in any copy number above the mean. Conversely there is only a finite number of copy numbers which the transcription factor can assume below the mean. Therefore at low enough means, one expects a slight deviation between the flux parameters. But these deviations are very slight as highlighted by the histograms and one can remain confident that the simulations reached the stationary state.

The steady state of the transcription factor is given when $\frac{d\langle x_1 \rangle}{dt} = 0 = \alpha - \langle x_1 \rangle / \tau_1$. Therefore the steady state mean of the transcription factor is $\langle x_1 \rangle = \alpha \cdot \tau_1$. Figure 8 shows with colour how when $\alpha \cdot \tau_1$ gets very small, the variance errors become very high due to the error propagation from the Gillespie $\langle x_1 \rangle$.

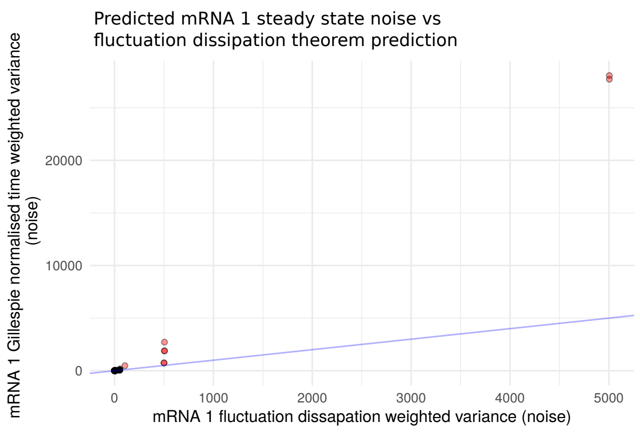


(a)

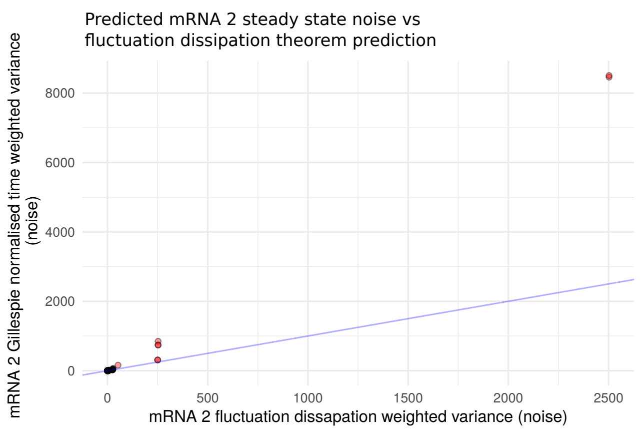


(b)

Figure 5: Cropped version of Figure 6 to illustrate that the vast majority of the simulations run were far enough from regimes where the choice of α and τ_1 could result in very large errors. Blue line is $y = x$, and all points lie on the line.

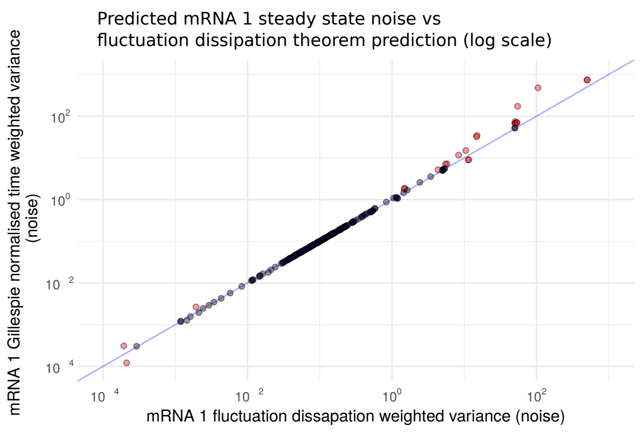


(a)

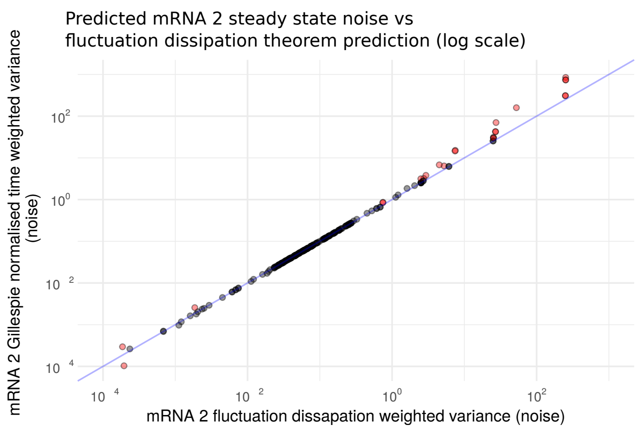


(b)

Figure 6: Plots which do not exclude high error scenarios, which occur when α is very low or when τ_1 is very low, resulting in a low steady state mean for the transcription factor. Points in red are those which had errors greater than 10%

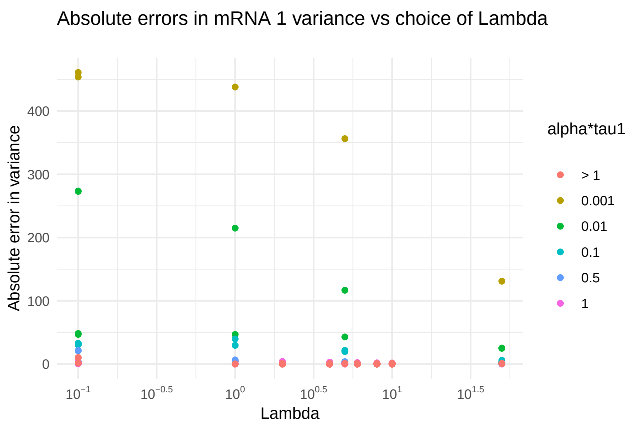


(a)

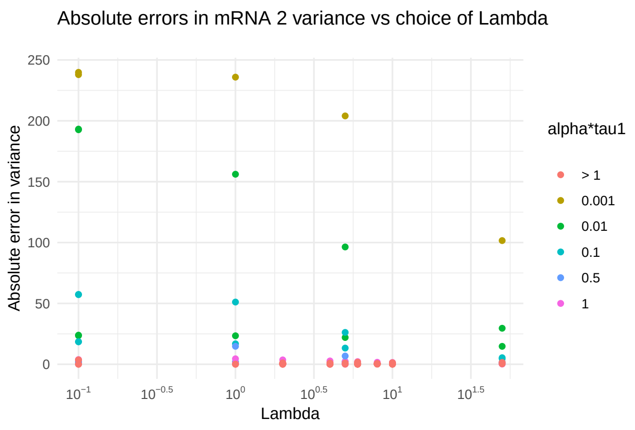


(b)

Figure 7: Figure 6 rescaled on log-log axes to capture all simulations. Points in red are those which had errors greater than 10%



(a)



(b)

Figure 8: Absolute errors between the FDT and the Gillespie algorithm for mRNA 1 and 2 variances with colour showing how decreasing steady state transcription factor mean causes proportionally higher errors.

Question IID

Traces of the transcription factor (orange) and the mRNA 2 (cyan) are shown for a baseline set of parameters ($\alpha = 10$, $\tau_1 = 2$, $\tau_2 = 2$, $\tau_3 = 4$ and $\lambda = 10$) and individual changes to those parameters. Each trace was created by running the Gillespie simulation for 1,000,000 iterations, and the figures display time slices (of length 40 thousand) from each trace.

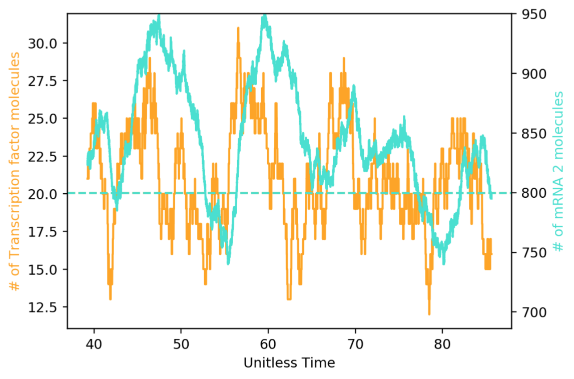


Figure 9: Baseline plot

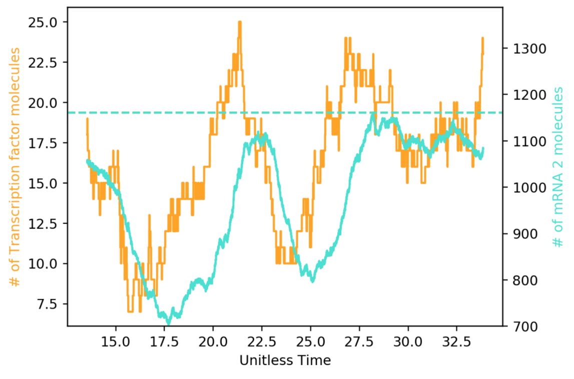


Figure 10: Baseline + high λ (=30)

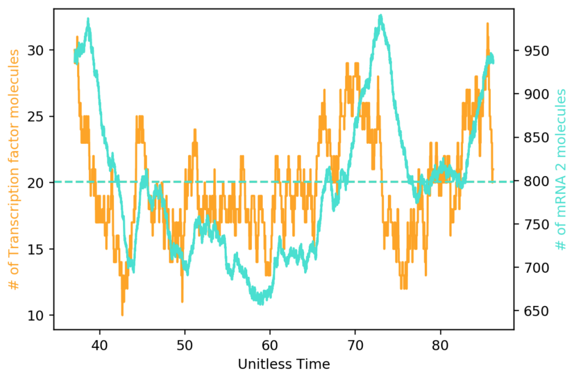


Figure 11: Baseline + low τ_2 and τ_3 (=1; 2)

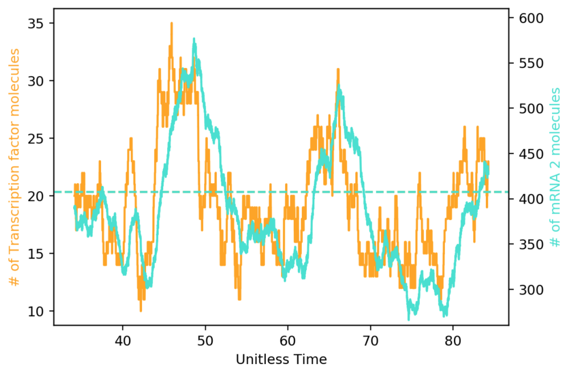


Figure 12: Baseline + low τ_2, τ_3 (=1;2) + high λ (=30)

As shown in Figures 9, 10, 11, 12, the coupling between the transcription factor and the mRNA levels changes depending on the parameter values. Compared to baseline, increasing λ decreased the lag during death events. Decreasing τ_1 and τ_2 instead decreases the lag during birth events. As is shown in Figure 12, having both a higher λ and a lower τ_2 and τ_3 leads to a tighter coupling between mRNA 2 and the transcription factor than in the baseline case.

Question IIE

Lastly, we looked at the dependency of the observed correlations of the mRNAs on the ratio of their CVs, Figure 13.

$$\rho_{23} = \frac{\eta_{23}}{\sqrt{\eta_{22}\eta_{33}}} \quad \text{and} \quad \mathrm{CV}_i = \sqrt{\eta_{ii}}$$

To better identify the resulting pattern, we produced two plots with the points coloured according to the used τ_1 or λ values. Additionally, we used different shapes for different α. We can see that with increasing τ_1 values, the ratio of the CVs increases, indicating that the normalised variance η_{33} increases in comparison to η_{22}. At the same time the correlations between the mRNAs increases with growing τ_1, going up to 0.9. This means that with longer lifetimes of the transcription factor, the abundances of the two mRNAs show very similar dynamics. Within this overall trend, we can see a second pattern being caused by λ. As λ increases the correlation of the mRNAs rises from below 0.6 to almost 0.85, while the CV ratio stays relatively constant. α on the other hand does not seem to have any effect.

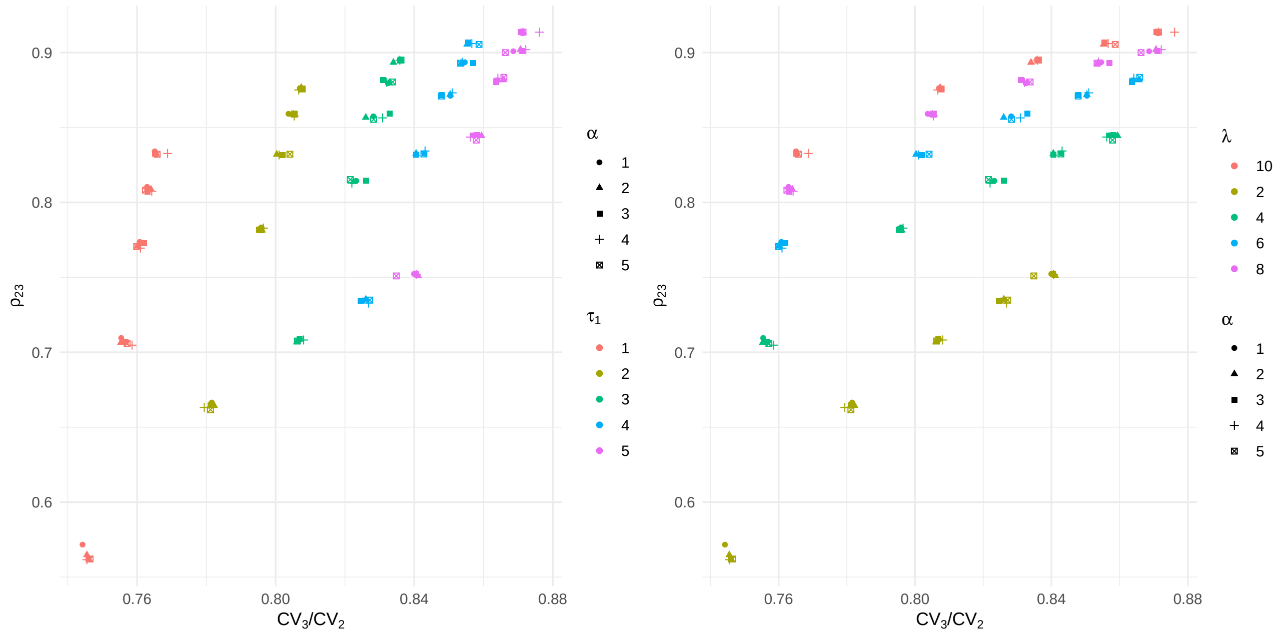


Figure 13: Observed correlation between the mRNAs versus the CV ratio.

Appendix

A Code

A.1 Model Definition and CLI

We simulate the model using the Gillespie algorithm. The model is defined as a Python class `mRNADynamicsModel`, and the Gillespie algorithm is optimised through just-in-time compilation (via Numba) to efficient machine code, speeding up the execution time by a factor of 40 compared to standard Python. A command line interface provides a convenient wrapper to run a single simulation (eg. `$ python gillespie.py --alpha=10 --tau1=2 --lambd=4 --iters=10000000`).

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as pyplot
from tabulate import tabulate # For text printing to terminal
from numba import njit  # For just-in-time compilation to LLVM
import random
import time
import click  # For command line interface


@njit # Numba decorator; compiles function to efficient machine code (LLVM)
def fast_gillespie(x, alpha, tau1, tau2, tau3, lambd, delta, N):
    """Optimised Gillespie algorithm using jit.

    Args:
        x (ndarray(int)): Initial counts for tf, mRNA1, and mRNA2
        alpha (float): transcription factor birth rate
        tau1 (float): transcription factor lifetime
        tau2 (float): mRNA 1 lifetime
        tau3 (float): mRNA 2 lifetime
        lambd (float): mRNA birth rate
        delta (ndarray): 2D diffusion matrix for system
        N (int): number of iterations for Gillespie

    returns:
        X (ndarray(int)): Trace of component counts for each iteration.
        T (ndarray(float)): Time adjusted trace of time during simulation.
        tsteps (ndarray(float)): Time weight trace; duration of time spent in each state.
    """
    # Initialisation
    t = 0
    T = np.zeros(N)
    tsteps = np.zeros(N)
    X = np.zeros((delta.shape[0], N))

    # Simulation
    for i in range(N):
        # Determine rates
        rates = np.array(
            [alpha, x[0] / tau1, lambd * x[0], x[1] / tau2, lambd * x[0], x[2] / tau3]
        )
        summed = np.sum(rates)

        # Determine WHEN state change occurs
        tau = (-1) / summed * np.log(random.random())
        t = t + tau
        T[i] = t
        tsteps[i] = tau

        # Determine WHICH reaction occurs with relative propabilities
        reac = np.sum(np.cumsum(np.true_divide(rates, summed)) < random.random())
        x = x + delta[:, reac]
        X[:, i] = x

    return X, T, tsteps


class mRNADynamicsModel:
    """Model for the dynamics of mRNA for two genes regulated by a transcription factor.

    Args:
        alpha (float): transcription factor birth rate
        tau1 (float): transcription factor lifetime
        tau2 (float): mRNA 1 lifetime
        tau3 (float): mRNA 2 lifetime
        lambd (float): mRNA birth rate
        delta (ndarray): 2D diffusion matrix for system

    Attributes:
        alpha (float): transcription factor birth rate
        tau1 (float): transcription factor lifetime
        tau2 (float): mRNA 1 lifetime
        tau3 (float): mRNA 2 lifetime
        lambd (float): mRNA birth rate
        delta (ndarray): 2D diffusion matrix for system
        labels (list(str)): labels for each component in system
    """

    def __init__(self, alpha, tau1, tau2, tau3, lambd, delta):
        self.alpha = alpha
        self.tau1 = tau1
        self.tau2 = tau2
        self.tau3 = tau3
        self.lambd = lambd
        self.delta = delta
        self.labels = ["T factor", "mRNA 1", "mRNA 2"]

    def compute_theoretical(self):
        """Computes theoretical means, variances, and covariances for system.

        returns: tuple of means, variances, and covar

        """
        # Theoretical states
        x1 = self.alpha * self.tau1
        x2 = self.lambd * self.tau2 * x1
        x3 = self.lambd * self.tau3 * x1

        # Theoretical variance and covariances
        v1 = 1 / x1
        v2 = 1 / x2 + self.tau1 / (x1 * (self.tau1 + self.tau2))
        v3 = 1 / x3 + self.tau1 / (x1 * (self.tau1 + self.tau3))

        cov12 = self.tau1 / (x1 * (self.tau1 + self.tau2))
        cov13 = self.tau1 / (x1 * (self.tau1 + self.tau3))
        cov23 = (
            self.tau1
            * (2 * self.tau2 * self.tau3 + self.tau1 * (self.tau2 + self.tau3))
        ) / (
            x1
            * (self.tau1 + self.tau2)
            * (self.tau1 + self.tau3)
            * (self.tau2 + self.tau3)
        )

        return [x1, x2, x3], [v1, v2, v3], [cov12, cov13, cov23]

    def run_gillespie(self, N=100_000_000, start_near_ss=False):
        """Runs Gillespie algorithm using model parameters.

        Note: Adds results from Gillespie as class attributes.

        """
        # Optional flag for starting simulation from predicted steady states when testing.
        # False by default.
        x = (
            self.compute_theoretical()[0] if start_near_ss else np.ones(3)
        )  # Start from one for each component.
        self.X, self.T, self.tsteps = fast_gillespie(
            np.ceil(x).astype(int),
            self.alpha,
            self.tau1,
            self.tau2,
            self.tau3,
            self.lambd,
            self.delta,
            N,
        )

    def get_fluxes(self):
        """Calculates difference in fluxes for each component.

        returns: List containing the flux balances for each component.

        """
        R1p = (
            (np.full((1, len(self.X[0])), self.alpha)[0]) * self.tsteps
        ).sum() / self.tsteps.sum()
        R1m = ((self.X[0] / self.tau1) * self.tsteps).sum() / self.tsteps.sum()
        R2p = ((self.lambd * self.X[0]) * self.tsteps).sum() / self.tsteps.sum()
        R2m = ((self.X[1] / self.tau2) * self.tsteps).sum() / self.tsteps.sum()
        R3p = R2p
        R3m = ((self.X[2] / self.tau3) * self.tsteps).sum() / self.tsteps.sum()
        return [R1p - R1m, R2p - R2m, R3p - R3m]

    def collect_stats(self):
        """Calculates statistics for most recent Gillespie simulation.

        returns: Tuple of means DataFrame & covars/vars DataFrame determined from Gillespie trace.

        """
        # Theoretical values
        Xss, Xvars, Xcovs = self.compute_theoretical()

        tw_means = (self.X * self.tsteps).sum(
            1
        ) / self.tsteps.sum()  # Time weighted means
        p_err = (Xss - tw_means) / Xss * 100
        fluxes = self.get_fluxes()
        mean_stats = pd.DataFrame(
            {
                "component": self.labels,
                "predicted_mean": Xss,
                "gillespie_mean": tw_means,
                "percent_error": p_err,
                "flux": fluxes,
            }
        )

        res = self.X - tw_means[:, np.newaxis]  # Residuals
        tw_vars = (self.tsteps * res ** 2).sum(
            1
        ) / self.tsteps.sum()  # Time weighted variances
        tw_w_vars = tw_vars / tw_means ** 2  # Weighted time-weighted variance
        combs = ((0, 1), (0, 2), (1, 2))  # Covariance combinations
        tw_w_covs = [
            ((self.tsteps * (res[c[0]]) * (res[c[1]])).sum() / np.sum(self.tsteps))
            / (tw_means[c[0]] * tw_means[c[1]])
            for c in combs
        ]  # Weighted time-weighted covariance

        labels = [f"var({lab})" for lab in self.labels] + [
            f"cov({self.labels[c[0]]}, {self.labels[c[1]]})" for c in combs
        ]
        fdts = np.array(Xvars + Xcovs)  # Concatenate lists for dataframe
        all_covs = np.array(list(tw_w_vars) + tw_w_covs)
        vars_covars_stats = pd.DataFrame(
            {
                "value": labels,
                "fdt": fdts,
                "gillespie": all_covs,
                "error": (fdts - all_covs) / fdts * 100,
            }
        )

        return mean_stats, vars_covars_stats

    def plot_X_trace(self):
        """Plots trace of Gillespie simulation for each component."""
        pyplot.plot(self.T, self.X[0], label="Transcription factor")
        pyplot.plot(self.T, self.X[1], label="mRNA 1")
        pyplot.plot(self.T, self.X[2], label="mRNA 2")
        pyplot.xlabel("Unitless Time")
        pyplot.ylabel("# of molecules")
        pyplot.legend(loc="best")

    def plot_flux_hist(self):
        """Plots flux histograms for each component of the Gillespie simulation."""
        d1, d2, d3 = self.get_fluxes()
        pyplot.hist(d1, 100, facecolor="red", alpha=0.5)
        pyplot.hist(d2, 100, facecolor="green", alpha=0.3)
        pyplot.hist(d3, 100, facecolor="blue", alpha=0.3)

    def __repr__(self):
        return f"<mRNADynamicsModel alpha: {self.alpha}, tau1: {self.tau1}, tau2: {self.tau2}, tau3: {s


@click.command()
@click.option("--alpha", default=10)
@click.option("--tau1", default=2)
@click.option("--tau2", default=2)
@click.option("--tau3", default=4)
@click.option("--lambd", default=4)
@click.option("--iters", default=100_000_000)
def main(alpha, tau1, tau2, tau3, lambd, iters):
    """Runs a single Gillespie simulation.

    Note: This func is intended as a CLI. All flags are optional.

    Args:
        alpha (float): transcription factor birth rate
```

```
            tau1 (float): transcription factor lifetime
            tau2 (float): mRNA 1 lifetime
            tau3 (float): mRNA 2 lifetime
            lambd (float): mRNA birth rate
            iters (int): number of desired iterations for Gillespie

        returns: Prints statistics for single run to terminal.
        """
        random.seed(42)
        D = np.array([[1, -1, 0, 0, 0, 0], [0, 0, 1, -1, 0, 0], [0, 0, 0, 0, 1, -1]])
        model = mRNADynamicsModel(
            alpha=alpha, tau1=tau1, tau2=tau2, tau3=tau3, lambd=lambd, delta=D
        )
        print(f"Initialised model: {model}")
        print(f"Running Gillespie algorithm for {iters:,} iters...\n")
        start = time.time()
        model.run_gillespie(iters)
        end = time.time()
        for table in model.collect_stats():
            print(tabulate(table, headers="keys", showindex=False), "\n")
        print(f"Gillespie took {((end-start)*1000):.2f} ms to complete.")


    if __name__ == "__main__":
        main()
```

A.2 One hundred simulations

Independent simulations are run in parallel using the Dask module.

```
import random
import numpy as np
import pandas as pd

# Require Dask for running independent simulations in parallel
from dask import compute, delayed
import dask.multiprocessing
from dask.diagnostics import ProgressBar

ProgressBar().register()

# Require model for multiple simulations
from gillespie import mRNADynamicsModel as M
```

```python


@delayed  # Used to delay execution until scheduled on thread by Dask
def gather_stats(model, tau1, alpha, lambd, N=100_000_000):
    """Gather statistics from Gillespie with different model parameters.

    Args:
        model: Instance of mRNADynamicsModel class
        tau1 (float): transcription factor lifetime
        alpha (float): transcription factor birth rate
        lambd (float): mRNA birth rate
        N (int): Number of Gillespie iterations (default=100_000_000)

    returns: Tuple of mean and var/covars DataFrames for Gillespie sim

    """
    # Adjust model parameters and run Gillespie
    model.tau1 = tau1
    model.alpha = alpha
    model.lambd = lambd
    model.run_gillespie(N)

    # Collect statistics
    means, covs = model.collect_stats()

    # Add columns to DataFrames for model params used in sim
    means["tau1"] = tau1
    means["alpha"] = alpha
    means["lambd"] = lambd
    covs["tau1"] = tau1
    covs["alpha"] = alpha
    covs["lambd"] = lambd

    return means, covs


if __name__ == "__main__":
    random.seed(42)

    # Diffusion matrix
    D = np.array([[1, -1, 0, 0, 0, 0], [0, 0, 1, -1, 0, 0], [0, 0, 0, 0, 1, -1]])

    # Intialise model
    model = M(alpha=10, tau1=2, tau2=2, tau3=4, lambd=10, delta=D)

    # Parameters to test
    taus = [1, 2, 3, 4, 5]
    alphas = [1, 2, 3, 4]
    lambds = [2, 4, 6, 8, 10]

    # Second test set
    # taus = [0.01, 0.1, 1, 5, 50]
    # alphas = [0.1, 1, 5, 10, 100]
    # lambds = [0.1, 1, 5, 50]

    # Create dask graph of desired simulations
    values = [
        gather_stats(model, t, a, lam) for t in taus for a in alphas for lam in lambds
    ]

    # Execute task graph, running independent simulations in parrallel via Dask
    results = compute(*values, scheduler="processes")

    # Concatonate means and covariances dataframes
    all_means = pd.concat([r[0] for r in results])
    all_covs = pd.concat([r[1] for r in results])

    # Write data to files
    all_means.to_csv("all_means.csv", index=False)
    all_covs.to_csv("all_covs.csv", index=False)
```

The code can be downloaded and tested at https://github.com/georgeoshardo/gillespy