May 23, 2017 – Hands On Intro to Simulation Stochastic Gene Expression
Prof. Joshua Weitz and QBioS Cohort

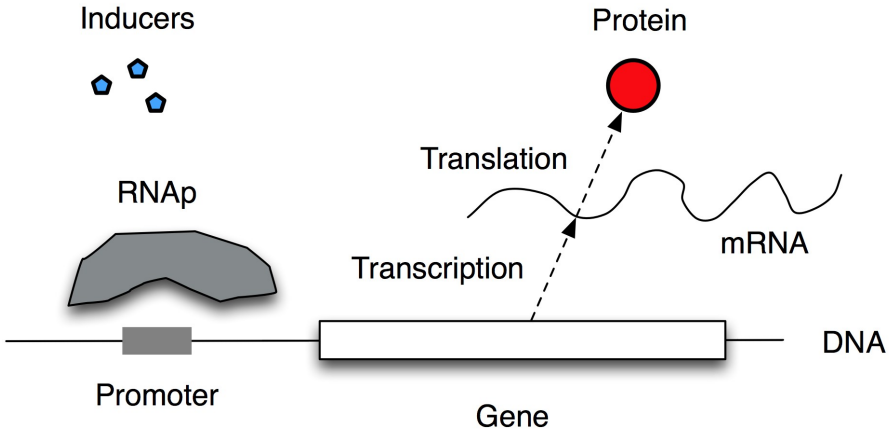
Would you like to be able to simulate stochastic gene expression? Building upon yesterday's hands-on intro to basic programming skills, we will get to the point where **you can do it**.

The goal of this lab is to understand how to:

- Translate gene expression processes into events
- Translate these events into a simulation
- Simulate stochastic gene expression that recapitulates expected dynamics and that can be compared to measurements.

The focal method of this lab is what is commonly known as the **Gillespie algorithm**. The Gillespie algorithm is a widely used approach developed for simulating chemical reaction kinetics, one event at a time. It can be used in many contexts, including stochastic gene expression.

Here then is our target. Consider a cell in which genes are transcribed and then the mRNA translated into protein. We will aggregate these two steps and say that proteins are produced at a rate β, e.g., 50 or 100 nM/hr. The units are important. For example, a single protein in the volume of an *E. coli* cell is approximately 1 nM concentration. Because cells are growing, we will assume that proteins decay at an exponential rate of 1 hr^{-1}. This oversimplifies the actual change in concentrations due to cell growth and division but will suffice, for now.



What will happen to protein concentrations given these two processes – protein production and decay? Will proteins decay away to nearly zero? Will concentrartions in cells go up and up, or will they reach a steady state? What exactly do we mean by steady state? Moreover, if production events are stochastic, as is decay, then how can we translate this highly simplified set of processes into a model? And, if we are able to do so for this case, is it possible that seemingly complex models might not be so complex to simulate after all? Enough questions. Let's get started. It'll be easier to explain if we start.

1 Time between events with a single process

1.1 Some theory

How long does it take for the next transcription to take place if the *rate* is b? This rate has units of inverse time, e.g., hrs^{-1}. There tends to be significant misunderstanding at what such units means. One way to think about them is that the inverse of the rate is a characteristic time, approximately equal to that of the time between random events. So events with a rate of 4 hrs^{-1} occur about every 15 minutes, whereas events with a rate of 10 hrs^{-1} occur about every 6 minutes. *Higher rates imply short intervals between events.*

Formally, this rate should be thought of as a *probability per unit time* that the event occurs. The probability that an event occurs in some small interval dt is $b \times \mathrm{dt}$. The probability that an event does not occur is $1 - b \times dt$. For some, you might recognize such events as arising from a Poisson process.

With rates defined, the question we want to ask is: *what is the probability that the event takes place at some time t from now.* For example, if an event occurs at a rate of 5 per hour, then what is the probability that the *next event* occurs precisely 0.2 hours from now, 0.4 hours, 2 hours, etc?

For an event to occur at time t for the first time, it should not occur anytime before-hand. This seems obvious, but it's the key to moving from rates to probabilities. We write $p(t)\mathrm{dt}$ to be the probability that the event occurs between time t and time $t + \mathrm{dt}$ where dt is some small time interval, e.g., 0.00001 hrs. Therefore, for the event to occur at time t, then it should satisfy the following equation:

$$p(t)\mathrm{dt} = \overbrace{(1 - b\mathrm{dt}) \ldots (1 - \mathrm{bdt})}^{\frac{\mathrm{t}}{\mathrm{Didnotoccurdt}} \text{times}} \cdot \overbrace{b\mathrm{dt}}^{\text{did occur}} \tag{1}$$

In words, this means that the event does not occur in any of the $\frac{t}{\mathrm{dt}}$ intervals between 0 and t. Then, the event does occur with probability $b\mathrm{dt}$ in the time interval between t and $t + \mathrm{dt}$.

Next, keep in mind that $b\mathrm{dt} \ll 1$, i.e., the probability is very small given small time intervals. Hence, we can approximate $1 - b\mathrm{dt} \approx \mathrm{e}^{-\mathrm{bdt}}$. Combining this together we find

$$\begin{aligned} p(t)\mathrm{dt} &= e^{-b\mathrm{dt}\frac{\mathrm{t}}{\mathrm{dt}}} \cdot b\mathrm{dt} \\ p(t)\mathrm{dt} &= e^{-bt} b\mathrm{dt} \end{aligned}$$

This is the key result. A Poisson process that occurs at a rate b leads to *exponentially* distributed events.

1.2 Some computing

There are multiple ways to generate exponentially distributed random numbers. The easiest is to use the built-in random number generator in MATLAB:

```
>> b=3;
>> exprnd(1/b)
```

Here, the rate is 3 per hr, and the function `exprnd` takes a single input – the inverse of the rate – and returns an exponentially distributed random number. Verify that this works by generating 1000 such random numbers and find their mean – fill in the missing part of the following commands:

```
>> tvals = exprnd(1/b,...);
>> meant = ...;
```

What do you expect the mean should be? In addition, plot the distribution of such random numbers, is it exponential? Now, change the rate parameter, e.g., setting it to 0.1, 10, or 2. Recall that the `hist` command is one way to visualize distributions, although there are other options as well.

Challenge problem: Attempt to generate exponentially distributed random numbers using only the uniform random number generator. Convert these samples to the time between two events in a Poisson process with a rate parameter $\lambda = 2\ \mathrm{hr}^{-1}$. **Hint:** Use the cdf of the exponential distribution, $cdf = 1 - e^{-\lambda t}$, to convert the uniformly distributed random number into an exponentially distributed random number. Repeat this 10000 times, what does the distribution look like?

1.3 Take-away

The take-away from this section is that when events happen at a random, each moment independent of the previous, then the time between events is exponentially distributed. This principles applies to protein production and decay, at least in this initial model.

2 Time between events with multiple processes

What happens if more than one kind of event can take place independently, each with their own characteristic rate? What then is the expected time before any event of either kind takes place?

In essence, we are asking: what is the time until process 1 occurs OR process 2 occurs. Let's consider two Poisson processes with rates $b_1 = 0.5$ and $b_2 = 2.5$. For the first event, sample two exponentially distributed random numbers using `exprand` (or using the conversion formula leveraging `rand` above).

```
>> b1=0.5;
>> b2=2.5;
>> r1=exprnd(1/b1);
>> r2=exprnd(1/b2);
>> etime1 = min(r1,r2);
```

The smaller time stored in `etime1` corresponds to the event that occurs first – this is the one that we record. The function `min` selects the minimum value amongst the two exponentially distributed random numbers. Repeat this procedure 10000 times, recording the times in an array. Here is one way to do it:

```
for i=1:1000,
  b1=0.5;
  b2=2.5;
  r1=exprnd(1/b1);
  r2=exprnd(1/b2);
  etime(i) = min(r1,r2);
end
```

Can you do better and vectorize this code? Now, what is the distribution of the variable `etime`, in other words the expected time to the first event? Use the `hist` command and you will notice that the events appear exponentially distributed. They are! The reason is that rates of events add. In other words, the next event occurs as if there was a single Poisson process with rate $b = b_1 + b_2 = 3$ per hour!

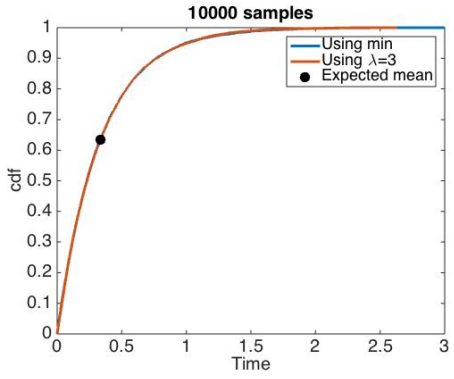
Formally, we can compare the distribution to the expected distribution from a Poisson process with $\lambda = 3$. When comparing continuous distributions it is often useful to compare the cumulative distributions. If the data of times between events is saved as a variable `etime`:

```
sortdata = sort(etime);
cdfvals = (1:length(sortdata))/length(sortdata);
plot(sortdata,cdfvals,'FontSize',20);
hold on
poiss3cdf = @(t) 1-exp(-3.*t);
plot(sortdata,poiss3cdf(sortdata),'FontSize',20);
hold off
set(gca,'FontSize',20)
```

This code has done a few things. Let's diagnose it, where I've added comments for added explanation.

```
% This sorts the event times from smallest to largest
sortdata = sort(etime);

% This determines the fraction of the data less thn a particular value
% This is the cumulative distribution
cdfvals = (1:length(sortdata))/length(sortdata);

% This next line creates a function, with one variable t that returns
% the CDF for an exponential distribution with rate parameter 3
poiss3cdf = @(t) 1-exp(-3.*t);

% This plots the CDF of the sorted times
```

```
plot(sortdata,cdfvals,'FontSize',20)

% This tells MATLAB to keep the current plot at the next command
hold on

% This plots the CDF of the predicted exponential distribution
plot(sortdata,poiss3cdf(sortdata),'FontSize',20)

% This tells MATLAB to remove the current plot at the next command
hold off

% This is just an example of how to change label axes sizes
set(gca,'FontSize',20)
```



Challenge problem: Can you figure out how many times each of the two processes took place first? What is the fraction of events associated with the rate b_1 and what fraction of events were associated with the rate b_2?

This section, and challenge problem, suggests an alternative method of simulating. Instead of choosing the shortest time, we could first independently sample the time according to a Poisson process with a rate corresponding to any process occurring. In general, the rate of any process occurring is $\lambda = \sum_j \lambda_j$, where λ_j is the rate of a single process, j and the sum includes all subprocesses. Next, we would sample which reaction occurs. The subprocess, j, is chosen with probability $\frac{\lambda_j}{\sum_j \lambda_j}$, where the sum includes all subprocesses. This approach is at the core of the Gillespie algorithm.

3 Gillespie algorithm applied to a gene expression model

3.1 Conceptual framework

n the previous section, we identified the times between events between two processes where the rates are constant. But rates can also depend on the state of the system. For example, bimolecular reactions occur at rates proportional the concentration of the two species involved:

- Substrate and enzyme
- Transcription factor and DNA binding site
- Receptor and ligand

Consider a cell in which proteins are produced at a constant rate β and decay at a rate α. The abundance of proteins, p, can be represented in terms of a differential equation:

$$\frac{dp}{dt} = \overbrace{\beta}^{\text{production}} - \overbrace{\alpha p}^{\text{decay}}. \tag{2}$$

units of inverse time. We have not yet discussed how to simulate such equations via the `ode45` command. Nonetheless, if one numerically integrates this equation given $\beta = 30$ hr^{-1} and $\alpha = 1$hr^{-1}, the system converges to a fixed amount. Even without integrating, it should be apparent that there is a protein abundance at which production balances decay. That balance happens when $p^* = \beta/\alpha$.

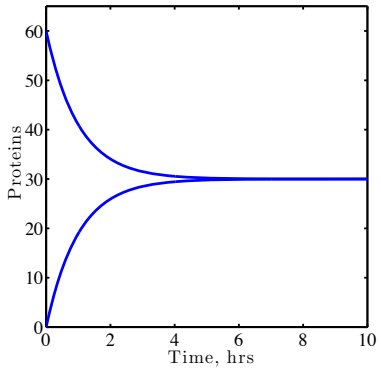


Figure 1: Dynamics of protein concentration given continuous model, given two initial concentrations, 0 and 60 proteins, and an equilibrium of 30.

But, is this what we expect to happen in a real cell? How can a stochastic algorithm capture the random nature of protein production? According to the Gillespie algorithm, the total rate at which any event occurs is the sum of the rates of individual processes:

Production Occurs at a constant rate β

Decay Occurs at a protein-dependent rate αp

Based on this, first try to sketch the probability the next event is a decay event as a function of the number of proteins. At equilibrium, $p^* = \frac{\beta}{\alpha} = \frac{30}{1}$. What is the probability of decay at equilibrium? Do the calculation by hand and reflect: does th answer seem intuitive?

Once you have made your sketch try to use Matlab to make the following figures:

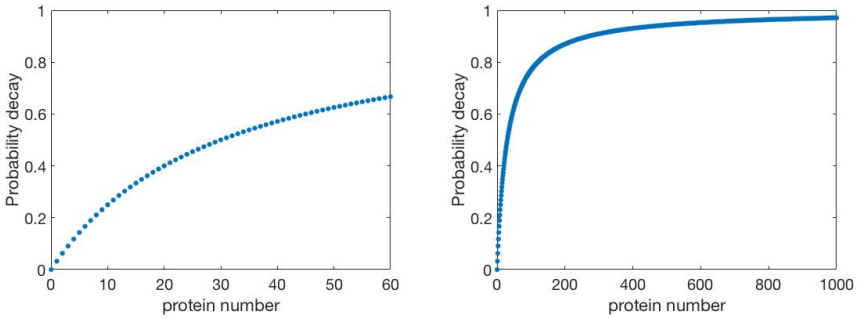


Figure 2: Probability that decay is the next event given a number of proteins. Probability of decay increases with protein number and limits to 1. (Left) Near the equilibrium protein number. (Right) Across larger number of protein to show approach to the limit.

3.2 Gillespie algorithm

In this example, we simulate stochastic trajectories of protein production where the decay rate depends on the total number of proteins (i.e., individual proteins have a constant rate of decay).

Before we simulate, it is worth discussing the algorithm in a generic form. Computer scientists often call this "pseudocode". Pseudocode is not written in a specific language but nonetheless lays out a series of steps that your actual code might follow. Here is a pseudocode version of the Gillespie algorithm:

```
specify time range of simulation
set initial time vector and protein numbers vector to 0
while the current time is less than the maximum
        update rate of decay based on number of proteins
        update rate of production
        find the waiting time until the next event
        update the current time based on the waiting time
        decide on which process occurred
        update and store the number of proteins based on selected event
        store the information about the system state at the current time
end
```

Discuss each step with your neighbor. Try to draw a sketch of what might happen given a cell that has initially 5 proteins vs. one in which there are initially 1 protein. Is every trajectory the same? Note that in this case, the rate of production remains constant irrespective of the current protein level. This need not be the case generally, e.g., as part of gene regulatory networks in which the current expression levels influence new production.

3.3 Implementing stochastic gene expression

Now we are ready to build a model of stochastic gene expression using the Gillespie algorithm. The following is one version, feel free to follow it or implement a version on your own!

Start a script, and enter the following (though the comments are optional):

```
% Part 1 - Initial conditions
p0 = 0; % Initial proteins
alpha = 1; % Decay rate
beta = 30; % Production rate
currp = p0; % Current proteins
tmax = 6; % Max time
currt = 0; % Current time
i=1; % Index
pvals(i)=currp; % Recording of proteins
```

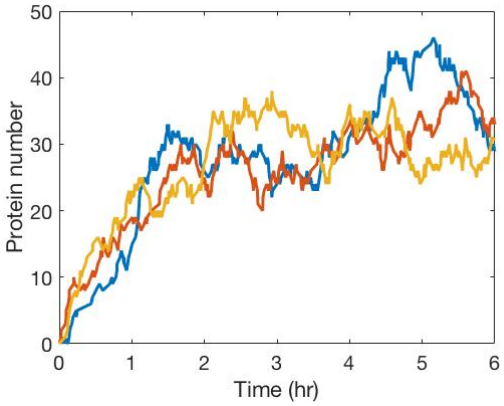
With this in hand, now start simulating:

```
% Part 2 - Stochastic simulation
while currt<tmax
  % Calculate rates
  decayrate = alpha.*currp;
  prodrate = beta;
  total_rate = decayrate+prodrate;

  % Find waiting time
  deltat = exprnd(1/total_rate);  % Find next event
  currt=currt+deltat; % Move forward in time
  prob_produce = prodrate/total_rate; % Calculate probability of production

  % Update the state
  if (rand<prob_produce) % Choose production at random
    currp = currp+1; % Increment number of proteins by 1
  else
```

```
    currp = currp-1; % Decrement number of proteins by 1
  end

  % Record the state
  i=i+1; % Increment the event counter
  pvals(i)=currp; % Store the protein number
end
```

That's it!

Plot a stochastic trajectory using the values stored in `pvals`. Three trajectories are plotted below to show some typical variation in the dynamics. The protein number approaches the expected state value of $p^* = \frac{\beta}{\alpha} = \frac{30}{1}$ and then appears to fluctuate about this value. The inherent noise from the sampling process causes the fluctuations around what would be an otherwise stable equilibrium for the analogous ODE dynamics.



Using this code, try out a few variations:

- Extend or shorten the maximum time
- Double the production, leaving the decay the same
- Double both the production and decay rates – does the mean change? If not, do you see any difference?
- ...

3.4 Bonus: discrete sampling of stochastic trajectories

Comparing stochastic trajectories across time can be difficult because the events occur at random times. It can be useful to sample the trajectory at fixed times. Write a function to sample a stochastic trajectory at fixed times. This pseduocode may be helpful:

```
Choose vector of times.
pre-allocate a vector the same size as the vector of times for sampled protein numbers
loop over vector of times
find the last protein number at the time less than or equal to the current time
place this value into the vector of sampled protein numbers
end
```

We are going to compare statistics of the stochastic trajectories across time and across trajectories. Run the gillespie algorithm with initially 0 proteins for a long time–say 200 hours. Use the last point as the initial condition for a new simulation. Run the simulation for another 1000 hours. Sample this trajectory between 200 and 1200 hours for 10001 equally spaced time points (this corresponds to sampling every 6 minutes).

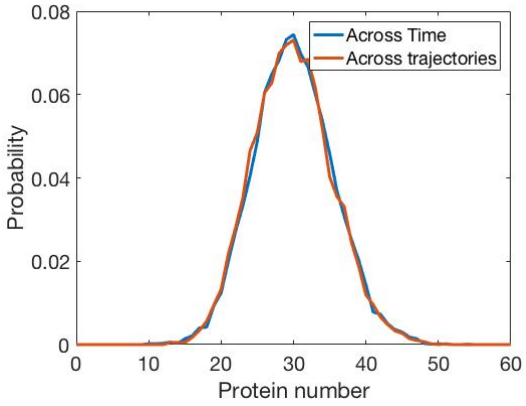


Figure 3: Poisson distribution of proteins at equilibrium in a stochastic gene expression model. Here the variation arises strictly from the fact that individuals production and decay events are random. The mean is the same as predicted from the continuous model.

What is the mean and variance of the protein numbers across time for this simulation? What is the distribution of protein numbers across time?

Now repeatedly simulate the dynamics for 200 hours starting at 0 protein number. Do this for 10001 runs (this should take less than a minute). For each simulation keep the final protein number.

What is the mean and variance of the final protein numbers across the trajectories? Compare this distribution of final protein numbers to the distribution across time that we computed above. Notice, the distributions are nearly the same. The notion that the state fluctuates across time in the same manner that it fluctuates across replicates is referred to as ergodicity. The main take-away of ergodicity is that if the dynamics are not expected to change significantly an experimentalist can choose to obtain statistics by either repeating an experiment or by increasing the number of samples of a trajectory. This is not expected to hold when transients are a major factor; hence, we simulated 200 hours after the initial condition before taking data.