An introduction to mathematical modelling for Systems Biology

Dr. Simon Rogers

November 22, 2013

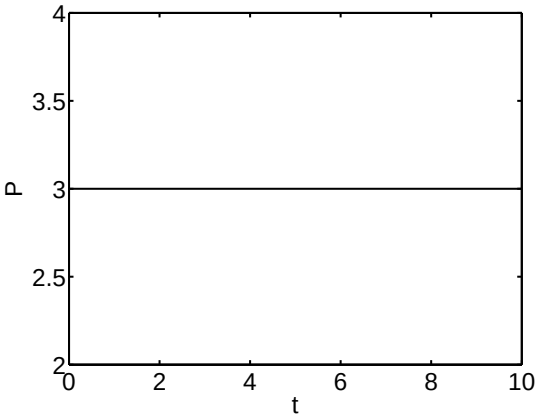
Contents

1 Introduction and disclaimer

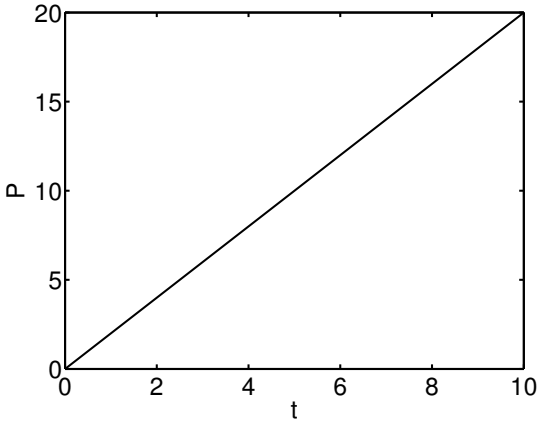
This is a *very short* introduction to some of the ideas behind modelling within Systems Biology. It is in no way exhaustive. It includes some mathematics, but not much. It is meant as a starting point for further study. Accompanying Java code to run the examples given in the text is available from Moodle (along with the files that define the examples). Instructions on how to compile and run this code can be found in the Appendices.

2 What is modelling? What is a model?

The term *model* crops up all over the place, with different meanings. In our context it is used to describe a *mathematical model* that tells us (in some sense) how the quantities of various biological quantities of interest change over time. Here are two simple examples, for some biological species (e.g. a particular type of protein) P:



(a) $P = 3$



(b) $P = 2t$

In the first example, P is constant (it doesn't depend on time, t). In the second it does, and increases as time increases.

2.1 Why model?

The purpose of building a mathematical model of a biological system is to formulate in either a mathematical or diagrammtical manner, a representation of our understanding of the system. This process is useful in itself (it pulls together knowledge of the system into a single place), but can also be used to *test* our understanding via making *predictions* of system behaviour that can be verified in the laboratory.

Building on our simple example above, our second model says that there will be 20 units of P at time $t = 10$. This is something that could be verified experimentally. If there are indeed 20 units then it provides some evidence that our knowledge is good (note that it doesn't prove it: many very different models could be built that would predict 20 units at $t = 10$, and they cannot all be realistic!). If there are not 20 units at $t = 10$, then there is something wrong with our current knowledge.

Note the careful choice of language in the previous paragraph: "...they cannot all be *realistic*." I could have said "...they cannot all be *right*." but it's important to realise that no model is every *right*. All models are built upon simplifying assumptions and can therefore never perfectly represent the world. This begs the question "how realistic does a model need to be?". And the answer to this depends very much on what you're hoping to achieve through the modelling process. We will examine the kind of assumptions that are made in more detail in the following sections.

2.2 Types of model

There are many types of model that are used in Systems Biology. Some involve time (as in our examples above) and describe how biological systems might *behave*, and some don't. For example, a graph (network) of protein-protein interactions would not typically model time (the network is assumed to be *static*). We will concentrate on those that do incorporate time and look at two broad families – differential equation models, and stochastic models. However, it is worth noting that there are many others. e.g.

- Systems of partial differential equations.

- Temporal logic.
- Petri nets.
- Process algebras.

2.3 Differential equations

The examples given above use equations that tell us directly how much P there is at some time t (in the first example, it is always 3, in the second it is always $2t$). Unfortunately, useful modelling of Biological systems is rarely so easy. In particular, it often makes more sense in biological systems to define how the quantities of different species are changing, rather than the quantities themselves. This is typically done by defining and solving *differential equations*.

We do not have time to go into much detail on differential equations here but I encourage you to read more. There are many introductory textbooks that cover these topics.

A differential equation looks something like this:

$$\frac{dP}{dt} = -\gamma P$$

The left hand side is read as "the rate of change of P with respect to time (t)" (although it looks like a division, it isn't!) and the right hand side defines what this rate of change is. Some people also use $\dot{P}$ as a shorthand for $\frac{dP}{dt}$.

The differential equations for our two examples above are:

$$\dot{P} = \frac{dP}{dt} = 0 \quad \text{and} \quad \dot{P} = \frac{dP}{dt} = 2,$$

respectively. In the first example, P never changes (it is always 3) and so its rate of change is 0. In the second its rate of change is 2 (it increases by 2 units for every unit of time).

If we try and re-create the plots above using just the rate of change information we come across a problem. In particular, $\dot{P} = 0$ just tells us that P doesn't change: we also need to know the starting value, $P_0 = 3$. Similarly, in the second example we need to know that $P_0 = 0$ to re-create the plot. These values are known as the *initial conditions*.

Let's now build a model of a more realistic but still simple biological example.

2.4 Example: protein decay

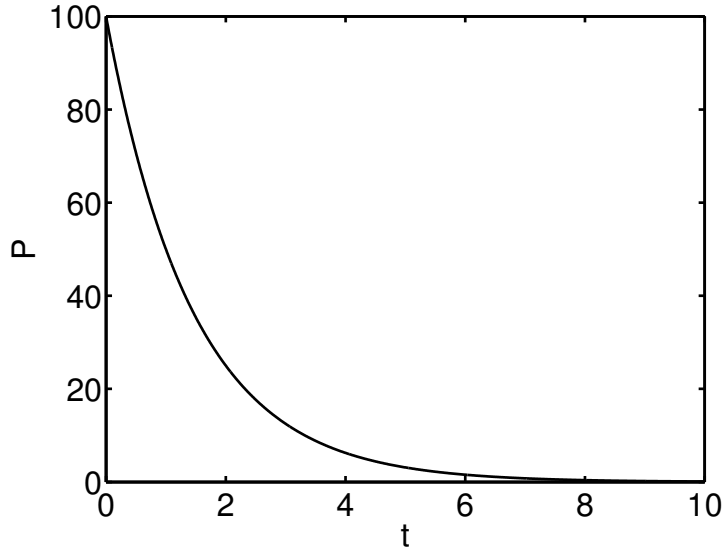
Consider a species of proteins (P) that 'live' for a while and then spontaneously decay. The proteins have a probability of 0.5 of decaying in a particular unit of time. In other words, for each currently suriving protein, we can toss a coin at the passing of each time unit and if the coin shows 'head', the protein decays. This suggests that after one time unit, roughly 50% of the proteins will have decayed. After the next time unit, roughly 50% of the proteins that survived the previous time step will have themselves decayed. Every time unit, the number of proteins will roughly half. This behaviour can be seen in the following Figure, where we start with a concentration of 100 proteins (note that we've moved to concentrations (i.e. proteins per unit volume rather than actual numbers – more on this later):

This is a common shape of function in the biological world (spontaneous decay is commonly assumed), and corresponds to the following differential equation:

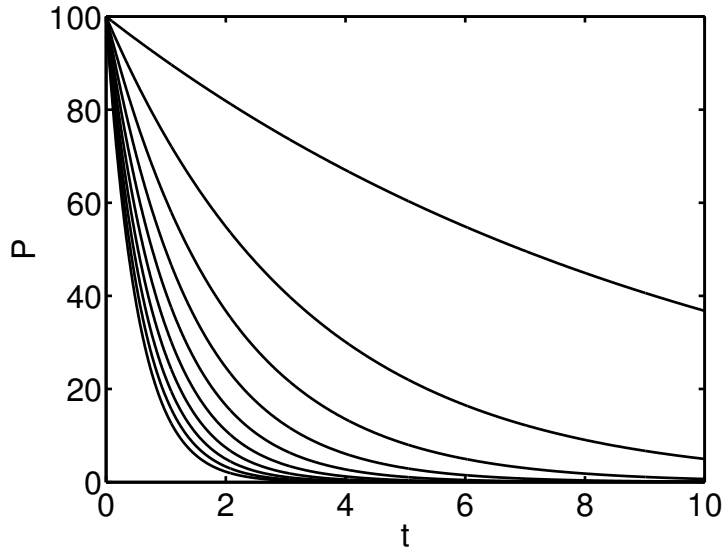
$$\dot{P}_t = -\lambda P_t$$

where P_t means "the value of P at time t" (remember P_0?) and λ is a *parameter* (in this case it is equal to $-\ln(0.5)$, but the particular value doesn't matter.)

Hopefully this equation makes some intuitive sense: $\dot{P}_t$ is always negative (the concentration of proteins is always *decreasing*) and it decreases faster the more P there is (the more times we toss the coin, the



more 'heads' we get). λ tells us how quickly P drops. The higher λ, the quicker the decrease. For example, the following plot shows the same curve for a range of λ values – the higest curve has $\lambda = 0.1$ and the lowest has $\lambda = 2$:



Note that λ is related to the probability of the coin showing heads, but isn't exactly the same.

Now, given the differential equation:

$$\dot{P}_t = -\lambda P_t$$

and an initial value, $P_0 = 100$, how do we actually get the plot above? We need the values of P_t and not the change in P_t. To translate from the latter to the former, we have to *solve* the differential equation. A very small proportion of differential equations that you will meet within Systems Biology can be solved analytically. That is, it is possible to write out an equation for P_t. This is one of those examples, and:

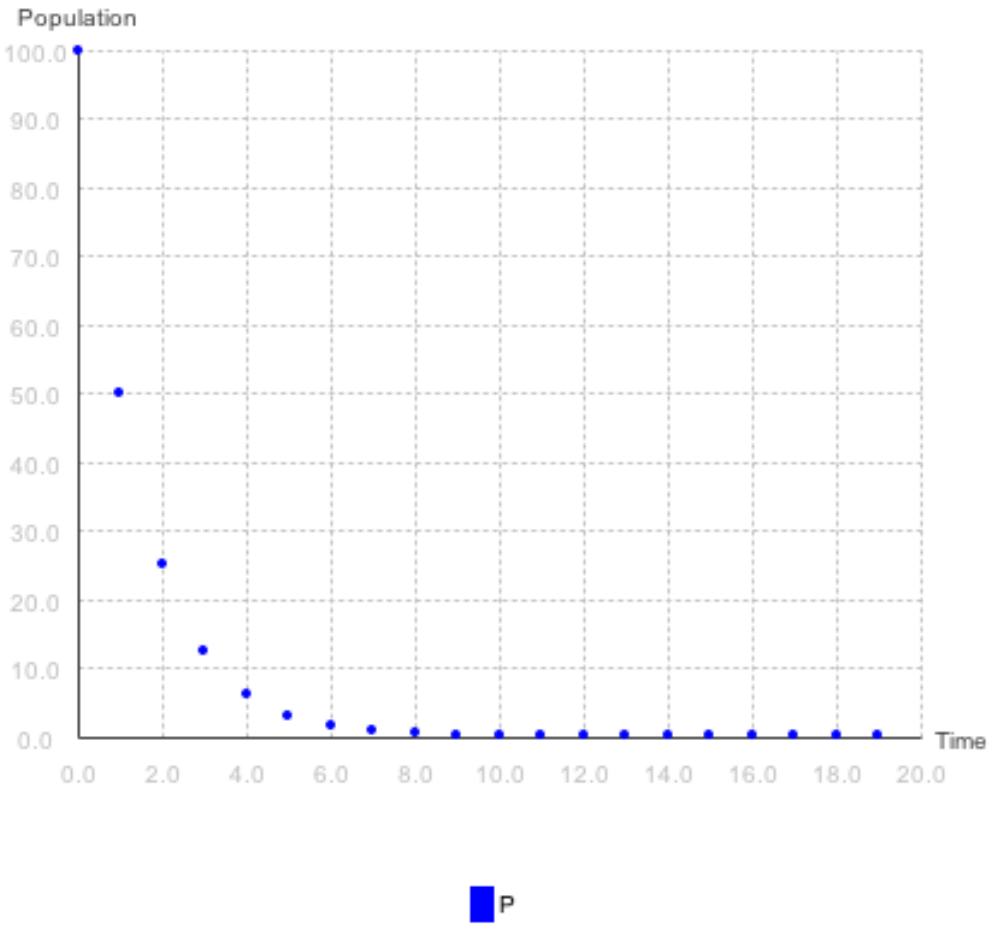
$$P_t = P_0 \exp(-\lambda t).$$

In general, we won't be so lucky and will have to resort to a numerical solver within a computer program of which there are many available. The Java code provided here is an example of the most simple method for numerically solving differential equations [more details in class; non-examinable]. It's important to note that this solver is *very bad* and should only be used for examples, and not any serious modellinga.

Using our solver, the results can be seen in the figure below. To generate this output, use:

```
>java DeterministicSolver decay 10 1e-2 1 out.txt
```

(see Appendices for explanation) and plot the results (`out.txt`) in your favourite plotting program.



3 Deterministic Modelling

The protein decay example in the previous section is an example of building a *Deterministic* model. The model is deterministic because when we solve the differential equation, we will always get the same result. This is a very strong assumption – biological systems aren't so well-behaved, but perhaps it's still a useful model?

In building this model, we also slipped in a few other assumptions:

- **Deterministic**. We have assumed that the system always behaves in the same way.
- **It makes sense to measure proteins as concentrations**. Implies that there are large quantities of the species.
- **There are no external influences**. The system is not influenced by anything else.
- **That the rate of decrease of protein concentration is proportional to protein concentration**.

We will discuss the limitations of these assumption in class, and later, when we consider stochastic simulations.

3.1 From reactions to equations

Our model will often be described by a set of reactions. For example our decay example is described by the following reaction:

$$\mathrm{P} \xrightarrow{\lambda} 0$$

Here's a more complex example:

$$\begin{aligned} \mathrm{A} + \mathrm{B} &\xrightarrow{k_1} \mathrm{A} + \mathrm{B} + \mathrm{B} \\ \mathrm{A} &\xrightarrow{k_2} 0 \end{aligned}$$

i.e., an A combined with a B creates an additional B in the first reaction. In the second, A decays. The *rates* of the two reactions are k_1 and k_2. These are analogous to λ in the previous example. To describe this system, we need two differential equations. One for A and one for B. A is involved in both reactions, so it stands to reason that both could potentially change A:

$$\dot{A}_t = \text{change caused by reaction 1} + \text{change caused by reaction 2}.$$

Reaction 2 looks like our protein decay example, so we'll use the same model we used there:

$$\dot{A}_t = \text{change caused by reaction 1} - k_2 A_t.$$

Inspecting reaction 1, how does A change? There is one molecule of A on each side, so there is, in fact, no change in A:

$$\dot{A}_t = 0 - \lambda A_t = -k_2 A_t$$

Moving to B, it is only involved in reaction 1, and when reaction 1 happens, a B is produced. We now have to make an additional modelling assumption – how do the quantities of A and B affect the rate at which this reaction happens? This most common assumption is *mass action*, where we assume that the rate at which the reaction occurs is proportional to $A \times B$ (mass action is actually a bit more involved than this, but it will do for our purposes). In other words, assuming mass action gives us the following differential equation for B:

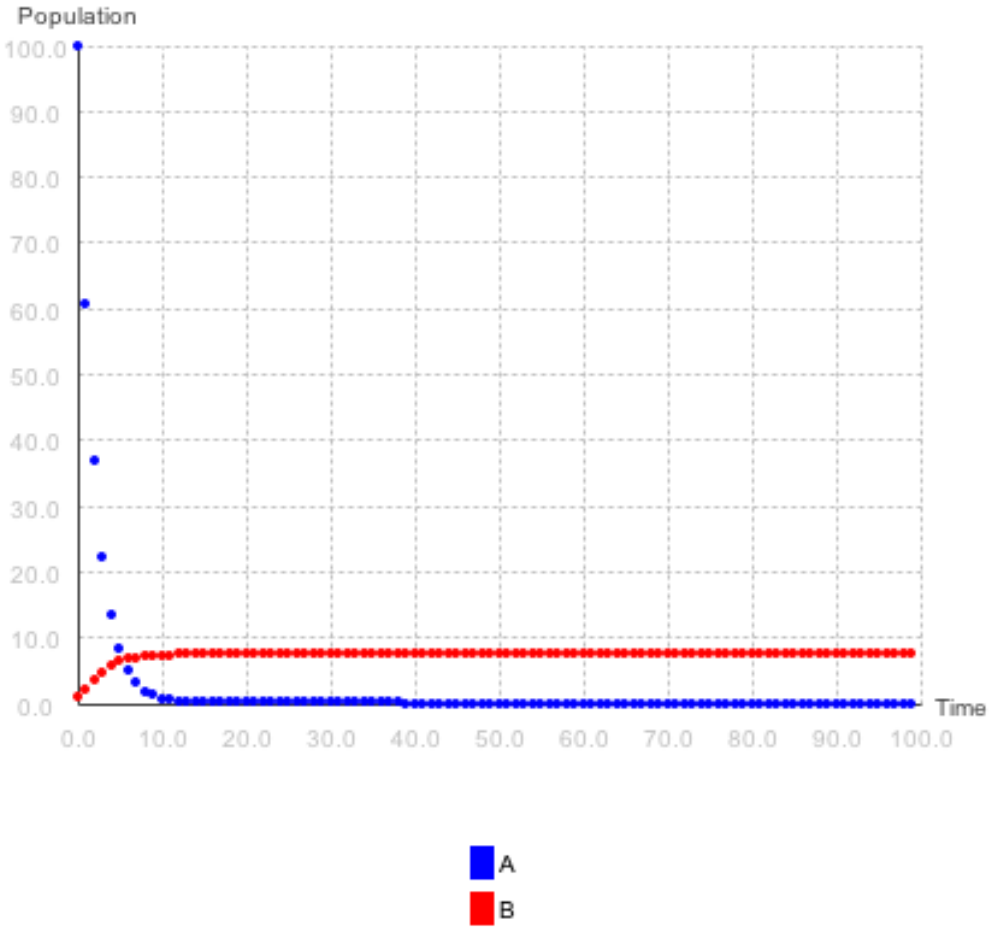
$$\dot{B}_t = k_1 AB$$

Our *system* of ODEs is therefore given by:

$$\begin{aligned} \dot{A}_t &= -k_2 A_t \\ \dot{B}_t &= k_1 A_t B_t \end{aligned}$$

To complete our definition, we need initial concentrations ($A = 100, B = 1$) and rates $k_1 = 0.01, k_2 = 0.5$. Running this model in our simulator gives the output shown in the Figure below. To produce this output, use:

```
>java DeterministicSolver maeg 100 1e-3 1 out.txt
```



Exercise: extend this model to also incorporate decay of B, with rate $k_3 = 0.01$. **Question:** what assumptions make up the mass action model?

3.2 Mass Action Example: Lotka-Voltera Model

Consider the following system of reactions:

$$\begin{aligned} \mathrm{A} + \mathrm{B} &\xrightarrow{k_1} \mathrm{A} + \mathrm{B} + \mathrm{B} \\ \mathrm{B} + \mathrm{C} &\xrightarrow{k_2} \mathrm{C} + \mathrm{C} \\ \mathrm{C} &\xrightarrow{k_3} 0 \end{aligned}$$

Taking the species in turn, and assuming mass action, we can derive a set of differential equations:

A : A doesn't change in any reaction, therefore:

$$\dot{A}_t = 0$$

B : B increases in reaction 1 (with rate $k_1 A_t B_t$) and decreases in reaction 2 (with rate $k_2 B_t C_t$). Therefore:

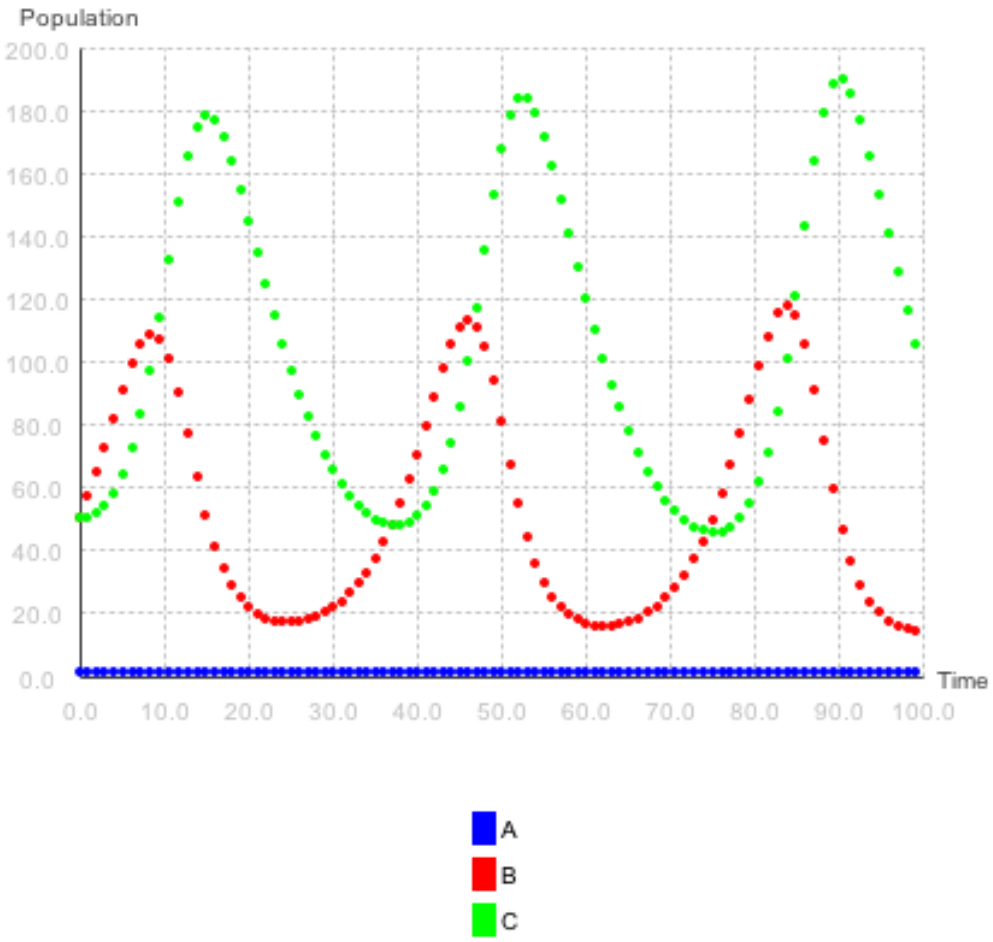
$$\dot{B}_t = k_1 A_t B_t - k_2 B_t C_t$$

C : C increases in reaction 2 (with rate $k_2 B_t C_t$) and decreases in reaction 3 (with rate $k_3 C_t$). Therefore:

$$\dot{C}_t = k_2 B_t C_t - k_3 C_t$$

Using the initial conditions $A_0 = 1, B_0 = 50, C_0 = 50$ and the rates $k_1 = 0.25, k_2 = 0.0025, k_3 = 0.125$, we obtain the following system behaviour when we solve the differential equations. To run this, use:

```
>java DeterministicSolver lotka 100 1e-4 1 out.txt
```



Notice that A remains constant (it has to: $\dot{A} = 0$), whilst the other two species oscillate. It's quite complex behaviour from what is a very simple set of reactions.

Exercise: vary the initial conditions and rates and observe the changes in behaviour.

3.3 Other kinetic models

In the previous example, we used the mass action kinetic model. Many others are used in biological models. A popular one is the Michaelis-Menten kinetic model. For example, for the following reaction:

$$\mathrm{A} \longrightarrow \mathrm{A} + \mathrm{B}$$

the differential equation governing B with mass action would be:

$$\dot{B}_t = kA_t$$

With Michaelis-Menten kinetics, it would be:

$$\dot{B}_t = \frac{k_1 A_t}{k_2 + A_t}$$

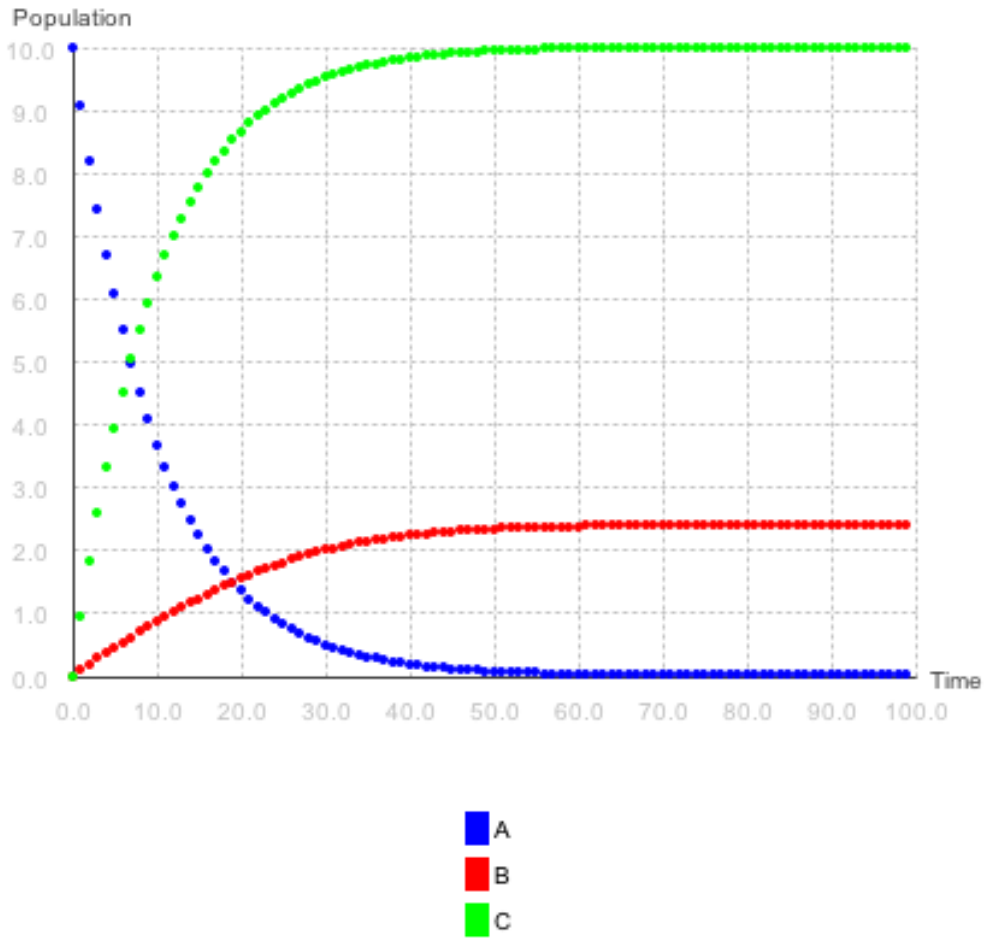
which requires two parameters: k_1 and k_2. In the example project `mmexample`, the following system of reactions is encoded:

$$\begin{aligned} \mathrm{A} &\xrightarrow{k_1} \mathrm{A} + \mathrm{B} \\ \mathrm{A} &\xrightarrow{k_2, k_3} \mathrm{A} + \mathrm{C} \\ \mathrm{A} &\xrightarrow{k_4} 0 \end{aligned}$$

where reactions 1 and 3 use mass action kinetics, and reaction 2 Michaelis-Menten. Using $A_0 = 100, B_0 = C_0 = 0$ and $k_1 = k_2 = 0.1, k_3 = 1, k_4 = 0.1$, we obtain the system behaviour shown in the next plot. To obtain these results, use:

```
>java DeterministicSolver mmexample 100 1e-3 1 out.txt
```

Exercise: Experiment with varying the values of k_2 and k_3. What is their role?


Population
10.0
9.0
8.0
7.0
6.0
5.0
4.0
3.0
2.0
1.0
0.0
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
100.0
Time
A
B
C

3.4 Practical exercise: gene regulation

1. The mRNA molecule mA is involved in the following two reactions:

$$0 \xrightarrow{b_{mA}} \text{A}$$
$$\text{A} \xrightarrow{d_{mA}} 0$$

   Write down the differential equation for $\dot{mA}$ (assuming mass action kinetics).

2. Write down the differential equation for $\dot{pA}$ assuming the mass action kinetics and the following reactions:

$$\text{mA} \xrightarrow{k_{mA}} \text{mA} + \text{pA}$$
$$\text{pA} \xrightarrow{d_{pA}} 0$$

3. Simulate this system using a Deterministic solver, with the following initial conditions and rates:

$$mA_0 = 0, pA_0 = 0 \qquad b_{mA} = 0.1, d_{mA} = 0.2, k_{mA} = 0.3, d_{pA} = 0.25$$

   Describe what happens.

4. (sightly more mathematical) Set your differential equation for $\dot{mA}$ to zero and solve for mA. Can you relate this solution to your model output? Do the same for $\dot{mP}$.

5. Add another two species to your model, mB and pB, with the following reactions:

$$0 \xrightarrow{b_{mB}} \text{mB}$$
$$\text{mB} \xrightarrow{d_{mB}} 0$$
$$\text{mB} \xrightarrow{k_{mB}} \text{mB} + \text{pB}$$
$$\text{pB} \xrightarrow{d_{pB}} 0$$
$$\text{pA} \xrightarrow{k_{pA}} \text{pA} + \text{mB}$$

   where all reactions are mass action. Use the following initial conditions: $mB_0 = pB_0 = 0$ and rates: $d_{mB} = 0.2, b_{mB} = 0.05, d_{pB} = 0.25, k_{mB} = 0.5, k_{pA} = 0.2$. Solve the system.

6. Change the last reaction to have MM kinetics with parameters 1,1. Experiment with these parameters and observe the change in the system.

7. Now change the final reaction to repression. Use the following kinetics (option 2 in the code):

$$\frac{V_{pA}}{pA + k_{pA}}$$

   And use the following initial conditions and rates: $mA_0 = 10, pA_0 = 2, mB_0 = 5, pB_0 = 0$, $b_{mA} = b_{mB} = 0.0, d_{mA} = d_{mB} = 0.2, d_{pA} = d_{pB} = 0.25, k_{mA} = 0.3, k_{mB} = 0.5, V_{pA} = k_{pA} = 1$.

8. (thinking) Why would a repression function that looked like $-kpA$ be bad?

9. (thinking) How could you make the model more detailed?

10. (thinking) How could you make the model less detailed?

11. (thinking) If you made predictions from this model, what kind of data could you generate to verify it? Does this have ramifications in how you might model?

4 Stochastic Modelling

In the first half of this short course, we looked at deterministic models and, in particular, models described by sets of differential equations. In this half, we will look at an alternative: Stochastic Simulation.

4.1 From concentrations to counts

In the deterministic models we worked with *concentrations* and we put no restriction on what values these concentrations could take. Consider a system where we have 1 molecule per unit volume. The decay model we built earlier would happily tell us that after one unit of time, we would have 0.5 molecules per unit volume. If we only have one unit volume in our system, this is nonsense – we can't have half a molecule! The deterministic modelling that we have done therefore looks decidedly dodgy when we have small numbers of molecules.

Stochastic simulation overcomes this problem by working with *integer* populations for each species. For the protein decay example, the state of the system at a particular time would be given by the exact number of proteins that are currently alive. Stochastic simulation moves through time by explicitly simulating each reaction in turn.

There are various algorithms around for doing this. We will use Gillespie's algorithm (paper available on Moodle).

4.2 Gillespie's algorithm

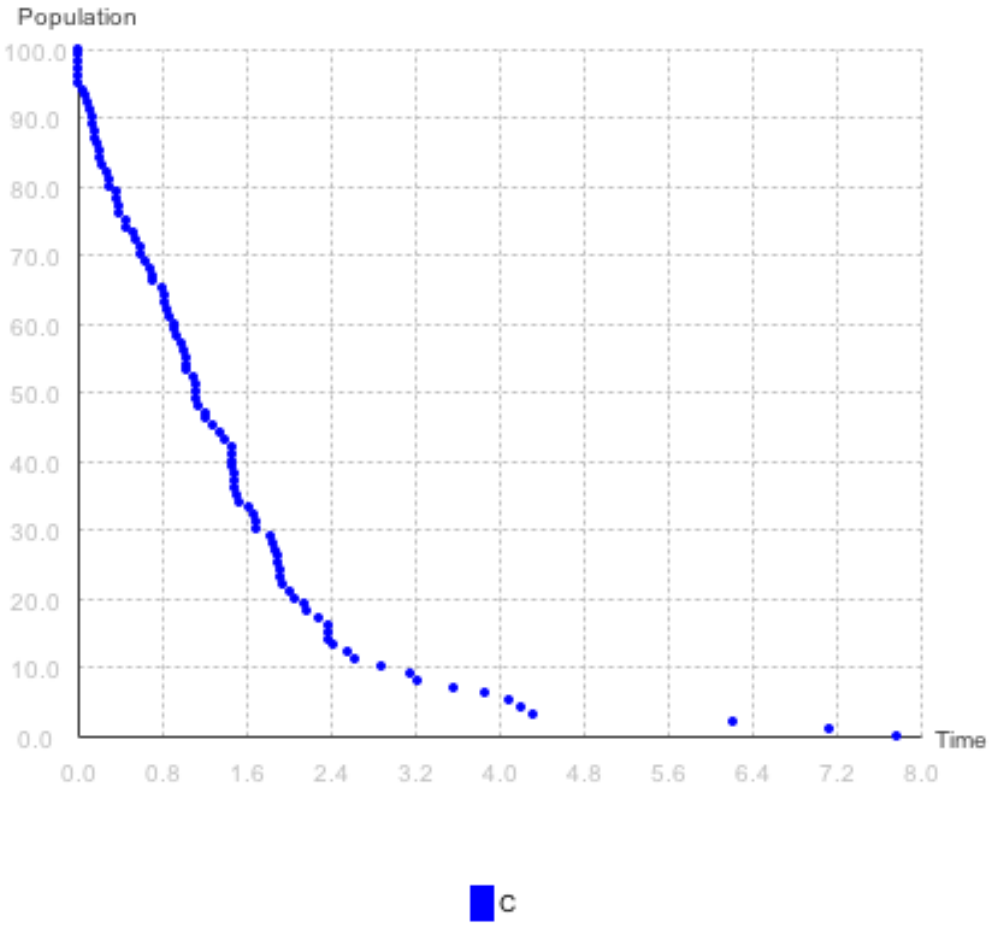
For some set of species and reactions, what Gillespie does is work out how long it will be until the next reaction, move forward this amount of time, choose which reaction should take place, update the species' populations and then repeat.

What makes Gillespie *stochastic* is that the choice of time and reaction are done *randomly*. That is, the Gillespie algorithm defines a *distribution* over the time and reaction from which particular instances are sampled.

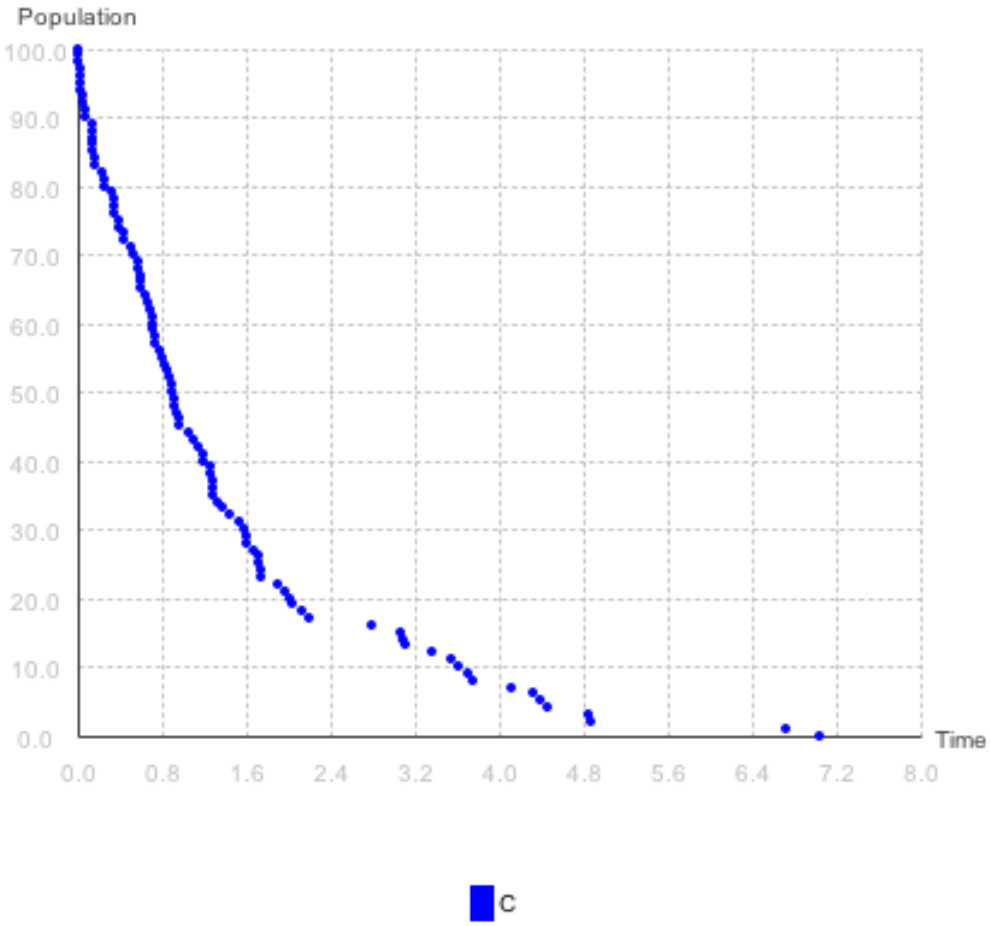
This is best seen with an example. Consider again our protein degradation. We start with, say, 100 proteins. Each protein has a degradation rate of λ which, in Gillespie's world means that the time until decay for each particular protein is *exponentially distributed* with parameter λ. You can think of this distribution as a black box – you set the dial to λ and it throws out times. As you change λ different times become more and less likely to be thrown out. So, staring at $t = 0$, we want to know how long it is until the first protein decays. Fortunately, because each of the individual times comes from the exponential box, we can get this time by setting our dial to 100λ (this is a property of the exponential distribution). So, we take a time from this box, move forward that time, subtract one from the number of proteins and then do it again, this time with the dial on 99λ, etc. When the number of proteins is zero, we stop.

If we do this, we get the following output:

```
>java GillespieSolver decay 200 out.txt 1
```


Population
100.0
90.0
80.0
70.0
60.0
50.0
40.0
30.0
20.0
10.0
0.0
0.0
0.8
1.6
2.4
3.2
4.0
4.8
5.6
6.4
7.2
8.0
Time
C

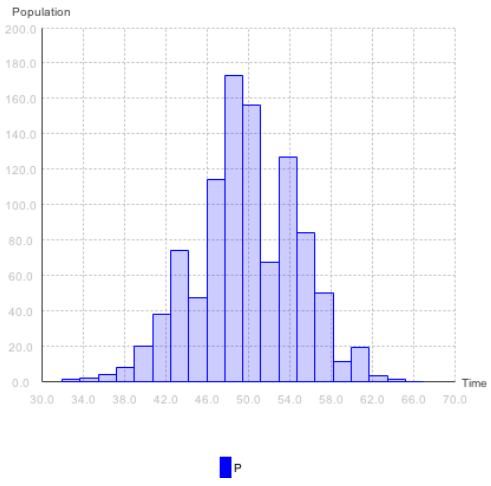
If we run it again, we get:



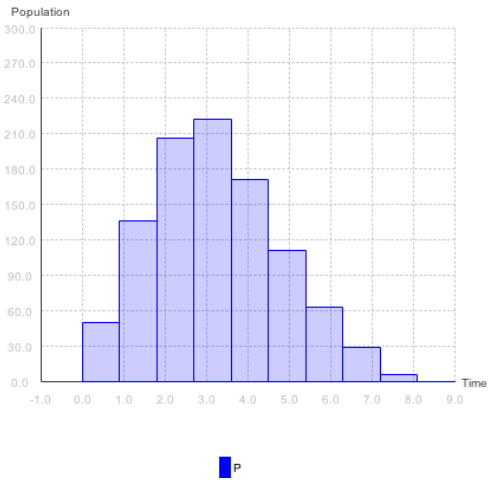
which is different! In fact, we can run it lots of times, and get a different result each time. This is because the model is stochastic: it gives us realisations of what could happen in reality. Gillespie is also known as *exact*: this is because it models populations exactly and generates trajectories that could happen in reality.

Obtaining different values each time it is run makes the output of the Gillespie algorithm slightly harder to interpret. Because of this, it is sometimes more useful to run the simulator many times and look at the spread of populations at particular times. For example, in the Figures below we can see the distribution of populations at $t = 1$ and $t = 5$ obtained from running the simulator 1000 times. To run the first one, do:

```
>java GillespieSolver decay 1 out.txt 1000
```


Population
200.0
180.0
160.0
140.0
120.0
100.0
80.0
60.0
40.0
20.0
0.0
30.0 34.0 38.0 42.0 46.0 50.0 54.0 58.0 62.0 66.0 70.0
Time
P

(c) $t = 1$


Population
300.0
270.0
240.0
210.0
180.0
150.0
120.0
90.0
60.0
30.0
0.0
-1.0 0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0
Time
P

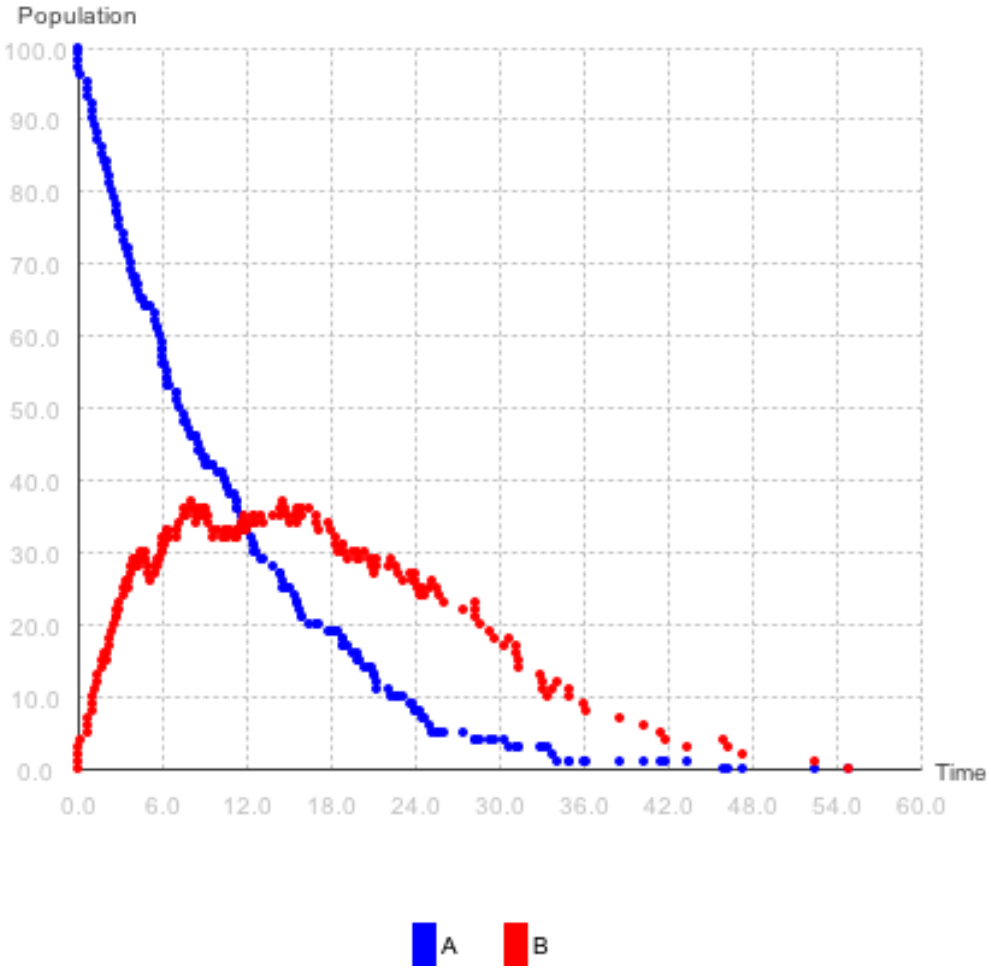
(d) $t = 5$

To fully explain the Gillespie algorithm, we need a system with more than one reaction. Consider the following set of reactions:

$$\begin{aligned} \mathrm{A} &\xrightarrow{k_1} \mathrm{B} \\ \mathrm{B} &\xrightarrow{k_2} 0 \end{aligned}$$

Using $A_0 = 100, B_0 = 1, k_1 = 0.1, k_2 = 0.1$, an example output from the Gillespie simulator is shown below. To run this, do:

```
>java GillespieSolver simple 100 out.txt 1
```



In this example, we have $N = 2$ species and $M = 2$ reactions. We will use X_n to denote the current count of the nth species. We will also use k_m to denote the rate of the mth reaction. Assume that we are currently at time t. The Gillespie algorithm does the following:

- For each reaction, m, compute $a_m = k_m h_m$. h_m is the number of reactant combinations, described below.
- Compute $a_0 = \sum_m a_m$
- Generate two uniform random variates, u_1 and u_2 (e.g. `Math.random()` in Java)

- Compute the time to the next reaction as $\tau = (1/a_0)\ln(1/u_1)$
- Increment t to $t + \tau$
- Choose the reaction v such that $\sum_m^{v-1} a_m/a_0 < u_2 \leq \sum_{m=1}^{v} a_m/a_0$ (this is a long-handed way of saying choose reaction m with probability a_m/a_0.
- Update the populations according to reaction v
- Return to 1.

h_m, is the number of reactant combinations - i.e. the number of combinations that can make up the left hand side of the reaction. For example, in the example above, the number of combinants for reactions one and two are A and B respectively. For the following reaction:

$$\mathrm{A} + \mathrm{B} \xrightarrow{k_1} \mathrm{C}$$

there are $A \times B$ ways of combining an A molecule with a B molecule. One slight complication is that reactions of the following type:

$$\mathrm{A} + \mathrm{A} \xrightarrow{k_1} \mathrm{B}$$

has $\frac{A(A-1)}{2}$ combinations.

And that's it – the Gillespie algorithm is very simple.

4.3 Practical exercise: Lotka Voltera with Gillespie

- Implement the model above, and plot histograms of the population at $t = 5$ based on 1000 runs.
- Take the Lotka-Voltera model described in the deterministic section and run it through the Gillespie solver. What do you notice? Explain what you see. What does it tell you about choice of modelling paradigms?
- To explore the limitations of the Gillespie simulator, implement the following model:

$$\mathrm{X} + \mathrm{X} \xrightarrow{k_1} \mathrm{Y}$$
$$\mathrm{Y} \xrightarrow{k_2} \mathrm{X} + \mathrm{X}$$

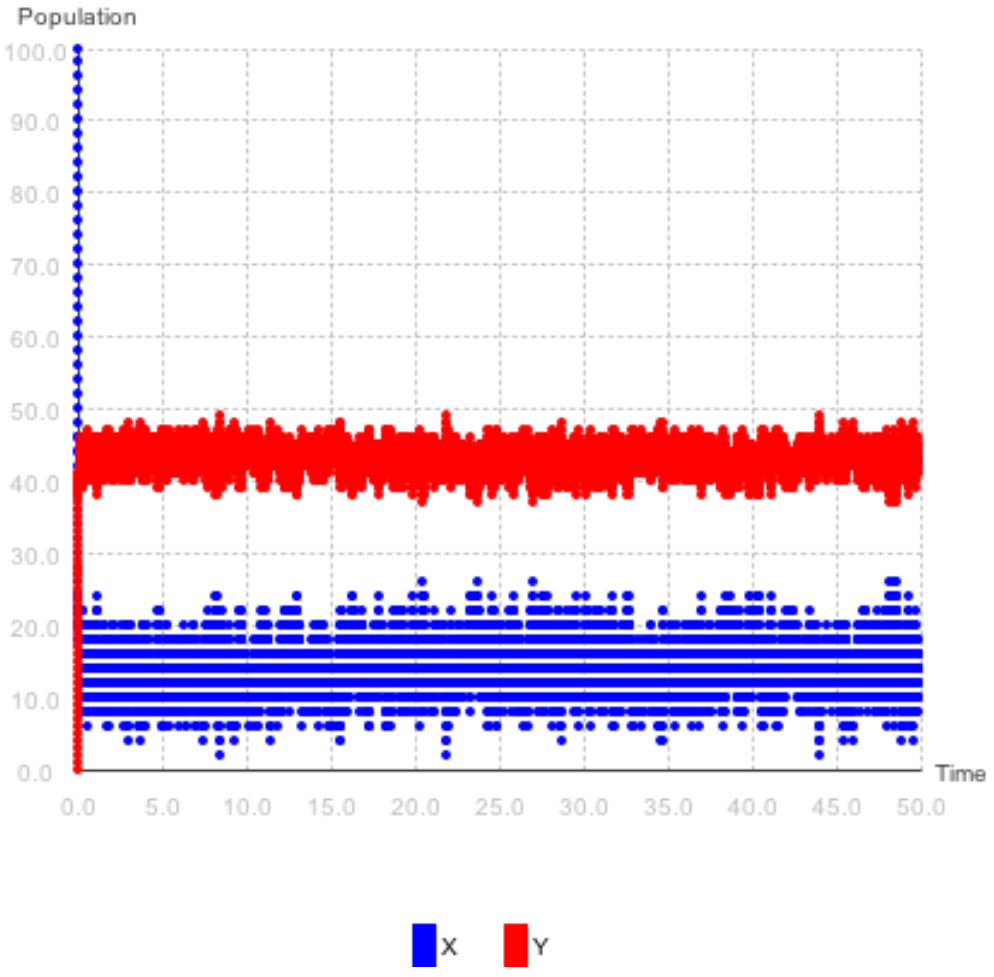
  Using $X_0 = 100, Y_0 = 1$ and $k_1 = 1, k_2 = 2$. Run the Gillespie simulator until $T = 50$. How many reactions are simulated? Plot the traces and also the histogram at $T = 20$. What do you notice? Is all this computational cost worthwhile? Try increasing the rates, what is the effect?

4.4 Speeding up Gillespie Simulators

As we've seen in the previous exercise, Gillespie can get **very** slow. There are two ways in which to make Gillespie faster:

1. Implement it very efficiently (can only get you so far).
2. Approximate it.

The latter option is popular and there are two key ways in doing this. Firstly, there is a technique known as τ-leaping. This involves jumping forward a certain amount of time, and working out how many reactions are likely to have happened in this time. The time step should be long enough to make an efficiency saving (i.e. ¿1 reaction should happen) but short enough that the state of the system shouldn't change *much* in this time. For example, the following plot shows the dimerisation example looked at at the end of the practical exercise. Throughout the simulation, there are ¿8000 reactions, but the state of the system doesn't change much. In this example, τ-leaping might be appropriate.
The second option is hybrid systems – model some parts of the system using a Gillespie system and some parts deterministically (see lecture slides for example).


Population
100.0
90.0
80.0
70.0
60.0
50.0
40.0
30.0
20.0
10.0
0.0
0.0
5.0
10.0
15.0
20.0
25.0
30.0
35.0
40.0
45.0
50.0
Time
X
Y

5 Data and Parameter Estimation

5.1 Data

Up to this point, we have taken quite an abstract look at mathematical modelling for Systems Biology, focusing on the mathematical techniques and not talking much about how one might use these things to 'do science'. We have discussed assumptions which are clearly important from a scientific view (you can't draw conclusions which take you outside the safety zone defined by your assumptions). A second key scientific point is how the model output can be compared with reality. There is little point in building a model, the output of which cannot be compared with measurements of the world. Conversely, this suggests that the measurements available have some bearing on the model construction and, in particular, the level of detail at which the system is to be modelled.

The stochastic simulation algorithm introduced in the previous section generates predictions of population counts. It can therefore be compared with data of this type (e.g. single cell protein count data derived from flourescence experiments). It may not be a suitable modelling paradigm if the only data available comes from complex mixtures of cells (e.g. DNA microarrays). On the other hand, a deterministic model may be appropriate for data from mixtures of cells (it perhaps conveys something about average cellular behaviour). I won't list more examples here, but it is a key point in model development: an intricate model is not much use if it cannot be compared with reality.

5.2 Parameter Estimation

In all of our examples, we have assumed that the parameters determining the rates of reactions were known. This is obviously not always the case and one goal of modelling might be to try and determine these parameters. Parameter estimation in Systems Biology is an open problem and one which (at the moment) is fairly feasible for (small) deterministic systems and fairly infeasible for (any) stochastic systems. I'll leave why this is to discuss in class.

All parameter estimation methods work in much the same way (although they can have quite different philosophies – more of this next semester). Basically, they guess some values for unknown parameters, solve the system, compare the system outputs with measured data and then guess some more values that hopefully make the match between the system and reality better. We will discuss this further in class if there is time.

Appendices

A Running Deterministic Java Solver

The deterministic java solver can be downloaded from Moodle2 `DeterministicSolver.java`. Move this file to a directory of your choosing and compile with:

```
>javac DeterministicSolver.java
```

To run the solver, you use the following syntax:

```
>java DeterministicSolver project tmax dt trecord outfile
```

The arguments are:

- `project` the project directory (more later).
- `tmax` time to simulate up to.
- `dt` time step for the solver (should be small!).
- `trecord` time step for recording results to the output file.
- `outfile` name of a file in which to record the results.

Alternatively, if you run the code with no command line arguments, you will be prompted to add the various parameters.

A.1 The project directory

The model is described in three files within the project directory. You can only have one model inside each project directory. The three files are `species.txt,init.txt,reactions.txt`:

- `species.txt`: a text file containing the names of the species. Each species should be on a new line.
- `init.txt`: a text file containing the initial concentrations for each species. They should be in the same order as the species in `species.txt` (terrible code!)
- `reactions.txt`: this is the meat of the project. It contains one line per reaction. The lines have the following syntax: `<reactiontype>:<lefthandspecies>:<righthandspecies>:parameters`. reactiontype is either 0,1 or 2 for mass action, MM, or repression respectively. lefthandspecies and righthandspecies are lists (separated by commas) of the species on the two sides of the reaction. Do not include any spaces, and make sure that the names are **identical** to those in `species.txt`. parameters are lists (separated by commas) of the parameters required by the reaction. For example, the following reaction:

  $$\mathrm{A} + \mathrm{B} \xrightarrow{k} \mathrm{A} + \mathrm{B} + \mathrm{B}$$

  using mass action with $k = 0.2$, would look like:
  `0:A,B:A,B,B:0.2`
  For reactions with 0 on either side, this field is just left blank. e.g. for

  $$\mathrm{A} \xrightarrow{k} 0$$

  we use (with $k = 0.2$):
  `0:A::0.2`
  For repression and Michaelis-Menten the parameter on the top of the fraction is given first and the two parameters are separated by commas. E.g.
  `1:A:A,B:0.2,0.3`
  corresponds to:

  $$\dot{B} = \frac{0.2A}{0.3 + A}$$

A.2 Output

The output file is tab-seperated. The first column gives the time, and the remaining columns the species concentrations. Each row is one time observation. The first row gives the species names. This data can be plotted in anything (e.g. gnuplot, excel, etc). If you would like to use my plotting code, see Appendix C.

B Running Stochastic Java Solver

The stochastic solver works in a very similar manner to the deteministic. In particular, the project structure is identical. To compile:

```
>javac GillespieSolver.java
```

And to run:

```
>java GillespieSolver project maxT outname nRep
```

The only variable that is different here is nRep. If nRep=1, the system works in the same manner as the Deterministic solver, placing the time series output in outname. If it is ¿1, the system runs the Gillespie nRep times and stores the output **at maxT** for each repetition as a row in outname. As with the deterministic solver, running with no arguments results in prompts for the necessary parameters.

C OutputPlotter

Also available on Moodle is my Plotter.java. The `build.sh` file will compile for you, or just use the `Plotter.jar`. If you provide no arguments, you will be prompted for the filename including the data you wish to plot, the type of plot (a time trace of the system, or a histogram of the values produced by running GillespieSolver with $nReps > 1$). If you choose histogram, you will also be asked for the number of bins. Note that due to a bug in jmathplot.jar, you cannot generate a histogram if all the data is the same, so you will not be able to plot a histogram of a constant species. To run from the command line:

```
>java -jar Plotter.jar outName trace
```

or

```
>java -jar Plotter.jar outName hist nBins
```