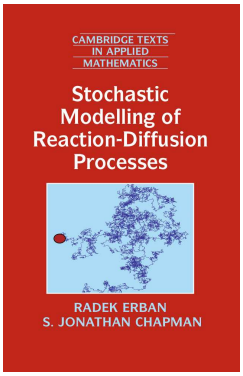
B5.1: Stochastic Modelling of Biological Processes

Lecturer: Prof. Radek Erban **Course Term:** Hilary 2021

Course Overview: "Stochastic Modelling of Biological Processes" provides an introduction to stochastic methods for modelling biological systems, covering a number of applications, ranging in size from molecular dynamics simulations of small biomolecules to stochastic modelling of groups of animals. The focus is on the underlying mathematics, i.e. it is not assumed that students have taken any advanced courses in biology or chemistry. The course discusses the essence of mathematical methods which appear (under different names) in a number of interdisciplinary scientific fields (including mathematical biology, non-equilibrium statistical physics, computational chemistry, soft condensed matter, physical chemistry or biophysics). New mathematical approaches and their analysis are explained using simple examples of biological models. The course starts with stochastic (non-spatial) modelling of chemical reactions, introducing stochastic simulation algorithms and mathematical methods which can be used for analysis of stochastic models. Different stochastic spatio-temporal models are then studied, including models of diffusion and stochastic reaction-diffusion modelling. The methods covered include molecular dynamics, Brownian dynamics, velocity jump processes and compartment-based (lattice-based) models.



Lecture Notes and Problem Sheets: The Lecture Notes for course B5.1 are published by the Cambridge University Press as

> R. Erban and S.J. Chapman, "Stochastic Modelling of Reaction-Diffusion Processes", Cambridge Texts in Applied Mathematics, CUP (2020)

The online version is available to everyone at all times through SOLO. College libraries also have physical copies. *Please read the recommended part of the Lecture Notes after watching each video lecture.* The course is accompanied by four Problem Sheets. Before attempting to solve Problem Sheet 1, please watch videos of Lectures $0, 1, 2, 3, 4, 5, 6$ and 7. Problem Sheet 2 covers Lectures 8–13, Problem Sheet 3 covers Lectures 14–17 and Problem Sheet 4 covers Lectures 18–21.

Prerequisites: This course builds on ten Prelims and Part A courses. Students taking this course should have mastered the material in the following courses:

> Prelims: Probability, Computational Mathematics, Introductory Calculus, Multivariable Calculus, Fourier Series and PDEs, and Constructive Mathematics
>
> Part A: Differential Equations 1, Probability, Integral Transforms and Complex Analysis

The Lecture Notes are self-contained and include some background material from above courses, but we will assume in our lectures and classes that students taking this course have a good understanding of the material covered in Prelims and Part A courses.

Computer Codes: Although some simple stochastic models can be analyzed using only a pen and paper approach, one cannot apply stochastic modelling to more complicated examples without using computers. Thus, our discussion is not only built around mathematical equations and their analysis, but also around the corresponding algorithms. The algorithms in the Lecture Notes are all written in a general pseudocode. Since the Prelims Computational Mathematics course introduced you to MATLAB, all computer-based exercises are designed in a way that they can be easily implemented in MATLAB. A brief refresher of MATLAB is included in Lecture 2 (the script for this lecture is provided on the next four pages). My computer codes which compute all illustrative simulations and figures presented in the Lecture Notes are available at: http://people.maths.ox.ac.uk/erban/cupbook/ However, you can use a programming language of your own choice to solve the computer-based exercises.

An extended script for Lecture 2, covering MATLAB and computer programming

The best way to learn stochastic simulation methods is to attempt to implement examples from the Lecture Notes and Problem Sheets using a computer language of your choice. One option is to use MATLAB which was introduced in Prelims Computational Mathematics. The choice of programming language is not important (you can use Fortran, C/C++, Julia or Python instead of MATLAB, if you prefer), but hands-on experience with implementing algorithms from the Lecture Notes in the computer can improve your understanding of the underlying mathematical methods.

In this note, I will show how you can design and write a MATLAB code which reproduces Figure 1.2(a) from the Lecture Notes. It will look relatively simple and straightforward. If you try to write your own computer code for Figure 1.2(a), then it might take you more time and your first result might be different than mine, but this is part of the learning process. If you do not get the answer which you expect, you have to go back to your computer code and break it into small components to find out which element was not correctly implemented. If you do not get the correct answer after a few back-and-forth iterations, then you can check the provided computer codes or ask your Class Tutor. However, please first try to identify yourself why your code does not work. By attempting it, you might actually fix the problem and learn more during this process.

Figure 1.2(a) shows results of five realizations of the SSA (a3)–(d3) together with the mean $M(t)$ (dashed line). Let us start with the dashed line. It is the solution of (1.15) with the initial condition $M(0) = 0$. This equation can be solved analytically to obtain

$$M(t) = \frac{k_2 \nu}{k_1} \left(1 - \exp(-k_1\, t)\right). \qquad (*)$$

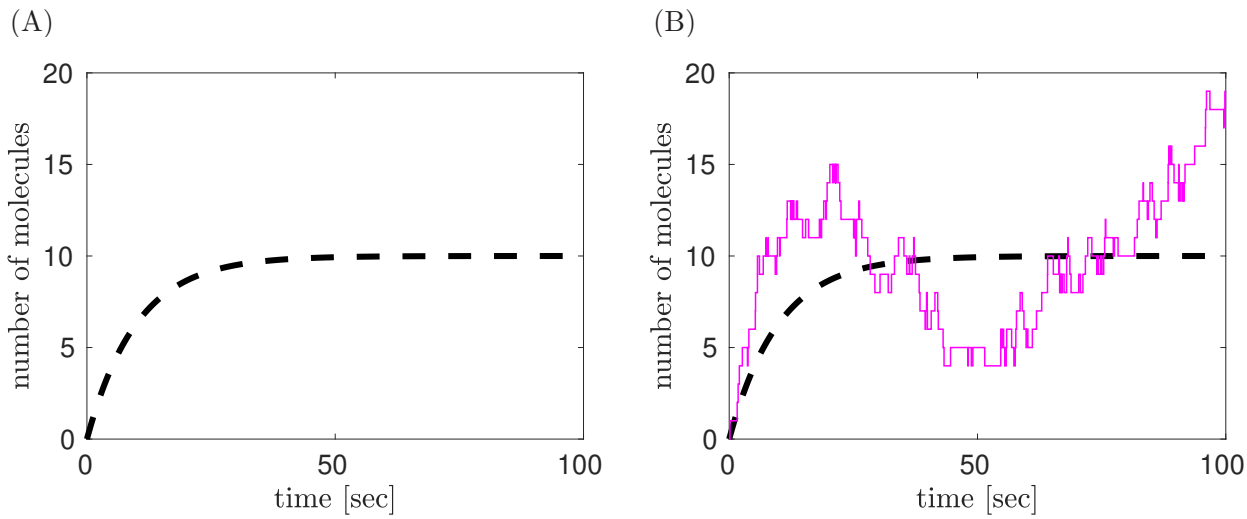
Therefore we can plot the time evolution of $M(t)$ using the following code in Matlab:

```
k1=0.1;
k2nu=1;
t=[0:0.2:100];
M=(k2nu/k1)*(1-exp(-k1*t));
plot(t,M);
```

The first two lines specify the values of parameters $k_1 = 0.1$ and $k_2\nu = 1$. The third line defines the values of time t: it is a vector with entries $[0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, \ldots, 99.6, 99.8, 100]$. The fourth line is the equation $(*)$ and the last line asks Matlab to plot M against time t. The above code correctly plots $M(t)$ but the resulting plot looks slightly different from Figure 1.2(a). To get Figure 1.2(a), I substituted the simple plotting command `plot(t,M)` with a few more lines to improve the output. The resulting code is:

```
k1=0.1;
k2nu=1;
t=[0:0.2:100];
M=(k2nu/k1)*(1-exp(-k1*t));
close all;
figure(1);
plot(t,M,'--k','Linewidth',4);
axis([0 100 0 20]);
xlabel('time [sec]','interpreter','latex');
ylabel('number of molecules','interpreter','latex');
```

The output given by this code is shown in the left panel on the next page (denoted panel (A)). The first four lines are the same as before and they do the actual computation. The fifth and sixth lines are not important to get panel (A), but I included them because you might find them useful later. The



(A) *Plot of* $(*)$ *given by the blue Matlab code in the text.* (B) *Plot of* $(*)$ *and one realization of the* SSA (a3)–(d3) *given by the combined (blue and red) Matlab code in the text.*

command `close all` on the fifth line will close all figure windows, if you have some figure windows open from the previous run of the code. The sixth line `figure(1)` specifies that we will plot into "Figure 1" (it is useful if the output of your code includes more than one figure). The plotting command on the seventh line has changed from simple `plot(t,M)` to `plot(t,M,'--k','Linewidth',4)`. It includes a few options, specifying that we will use a black dashed line and its width. The eighth line says that we will plot the interval $[0, 100]$ on the time axis and the range of M will be interval $[0, 20]$. Finally, the last two lines specify labels on each axis.

Next, we implement the SSA (a3)–(d3) in Matlab. We start by initializing our main variables `A` (number of molecules) and `time`. We also introduce vectors `Aplot` and `timeplot` which will be used for storing and plotting our results. Thus the initialization part of our code is:

```
A=0;
time=0;
Aplot=A;
timeplot=time;
```

We need to sample random numbers which are uniformly distributed in $(0, 1)$. In Matlab, we can use the command `rand(2,1)` which gives us a vector of two random numbers `r(1)` and `r(2)` which are needed during each iteration of the SSA (a3)–(d3). It is good practice to initialize the random number generator. Matlab uses the following command:

```
rand('state',5);
```

Then we get the same sequence of "random numbers" whenever we run the Matlab code. This is very helpful whenever you try to identify errors in your computer code which might otherwise be difficult to find. We add the red lines after the blue lines. In particular, the values of constants k_1 and $k_2\nu$ are already specified in the blue part of the Matlab code. In each iteration, we will store the number of molecules and the corresponding time in vectors `Aplot` and `timeplot`. Typing `Aplot=[Aplot A]` or `timeplot=[timeplot time]`, Matlab adds the current value of `A` or `time` as the last entry of the vector where we store our results. This construction is then used in the main loop of our computer implementation of the SSA (a3)–(d3) which is given on the next page:

```
while (time<100)
      r=rand(2,1);
      a0=k1*A+k2nu;
      time=time+(1/a0)*log(1/r(1));
      if (r(2)*a0<k1*A)
          A=A-1;
      else
          A=A+1;
      end
      Aplot=[Aplot A];
      timeplot=[timeplot time];
end
```

The results stored in vectors `Aplot` and `timeplot` are plotted using the following commands.

```
hold on;
h=stairs(timeplot,Aplot);
set(h,'Color','m','Linewidth',1);
```

where the first command `hold on` helps us to plot the computed realization of the SSA (a3)–(d3) into the same figure as the result of the blue part of our code. To plot `Aplot` against `timeplot`, we use the command `stairs` instead of `plot`. This helps use to emphasize that the value of A stays constant until a reaction occurs. The result computed by the combined blue and red codes is plotted in panel (B).

Finally, we can obtain five realizations as plotted in Figure 1.2(a) by using the above code five times. We have to make sure that we use different random numbers to get different time evolutions of `A`. The resulting Matlab code which plots Figure 1.2(a) is provided as `Figure1_2a.m` on the following website:

http://people.maths.ox.ac.uk/erban/cupbook/

As the next step, you could try to compute the stationary distribution presented in Figure 1.2(b). You could run many realizations of the above red computer code to obtain the presented histogram. Please note that you will obtain more efficient codes in Matlab if you only store statistics which you are interested to plot (histogram), i.e. do not introduce `Aplot` and `timeplot` to store details of all computed realizations. Otherwise, you could end up with a slower code (if you attempt to store and work with unnecessary large data files). If you have any difficulties, you can also check the Matlab code `Figure1_2b.m` on the above website.

If "Stochastic Modelling of Biological Processes" was being taught to students of biology, then the emphasis would be on algorithms and their implementations. They would be focussing on solving complex biological problems using the algorithms from the Lecture Notes. However, the course B5.1 is offered in the Mathematical Institute and it is not only about programming. Our aim is to understand the mathematics which can be found behind the algorithms from the Lecture Notes. Therefore, once you have a code for the corresponding figure in the Lecture Notes, your next question should be whether this code gives you the answer which you expected.

For all problems (figures) in the Lecture Notes, it is easy to know what you should expect: The goal of your simulations is to get the same answer (figure) as it is presented in the Lecture Notes. However, one can also write suitable mathematical equations and get the same results without doing any simulations, if the problem is reasonably simple. For example, in Figure 1.2(b), you can see the red line which can be obtained analytically using formula (1.24) or by solving equations (1.20)–(1.21).

Obviously, in real complex biological problems, you do not know a priori the answer which you should expect from your model or computer code. You can replace a complex biological model (which you

do not fully understand) by a complex computer code. Such a computer code can give you interesting (counterintuitive) graphs, but you need some confidence that your computer code is doing what it should do. One option is to break your large code into small subsystems which you fully understand, i.e. small subsystems which you can analytically solve. Course B5.1 studies such simple models. In fact, two chemical reactions which form the underlying model behind Figure 1.2 can be found as a simple subsystem in larger biological models.

The figures in the Lecture Notes often show that the same information can be obtained from the analysis of suitable differential equations as from stochastic simulations. In particular, there is no need to do simulations for many examples. However, teaching this course without discussing stochastic simulation algorithms could give you the wrong impression that "Stochastic Modelling of Biological Processes" is a computer-free subject. In the Lecture Notes, we will focus on simple systems where some analysis is possible. Complex biological models can still be analyzed by writing computer codes, but their mathematical analysis is complicated or even impossible (unless we start to use some simplifying assumptions). However, even the computer-assisted analysis can be limited, because the resulting simulations can be computationally intensive. This is another area where mathematicians can usefully contribute. Chapters 5 and 9 of the Lecture Notes include a discussion of a number of methods which can be used when computer codes are very slow. Chapters 5 and 9 and non-examinable.

My computer codes which compute all illustrative simulations and figures presented in the Lecture Notes are available online at:

http://people.maths.ox.ac.uk/erban/cupbook/

Some of these codes are written in Fortran to increase the efficiency of simulations. However, you are not expected to learn Fortran or any other new computer language for the purposes of this course. You can write all your own codes in MATLAB, which was covered in the Prelims Computational Mathematics course. All essential codes have been rewritten to MATLAB on the above website.