COMPANION IN MATLAB

QUANTITATIVE BIOSCIENCES

Dynamics across Cells, Organisms, and Populations

JOSHUA S. WEITZ
BRADFORD P. TAYLOR

QUANTITATIVE BIOSCIENCES COMPANION IN MATLAB

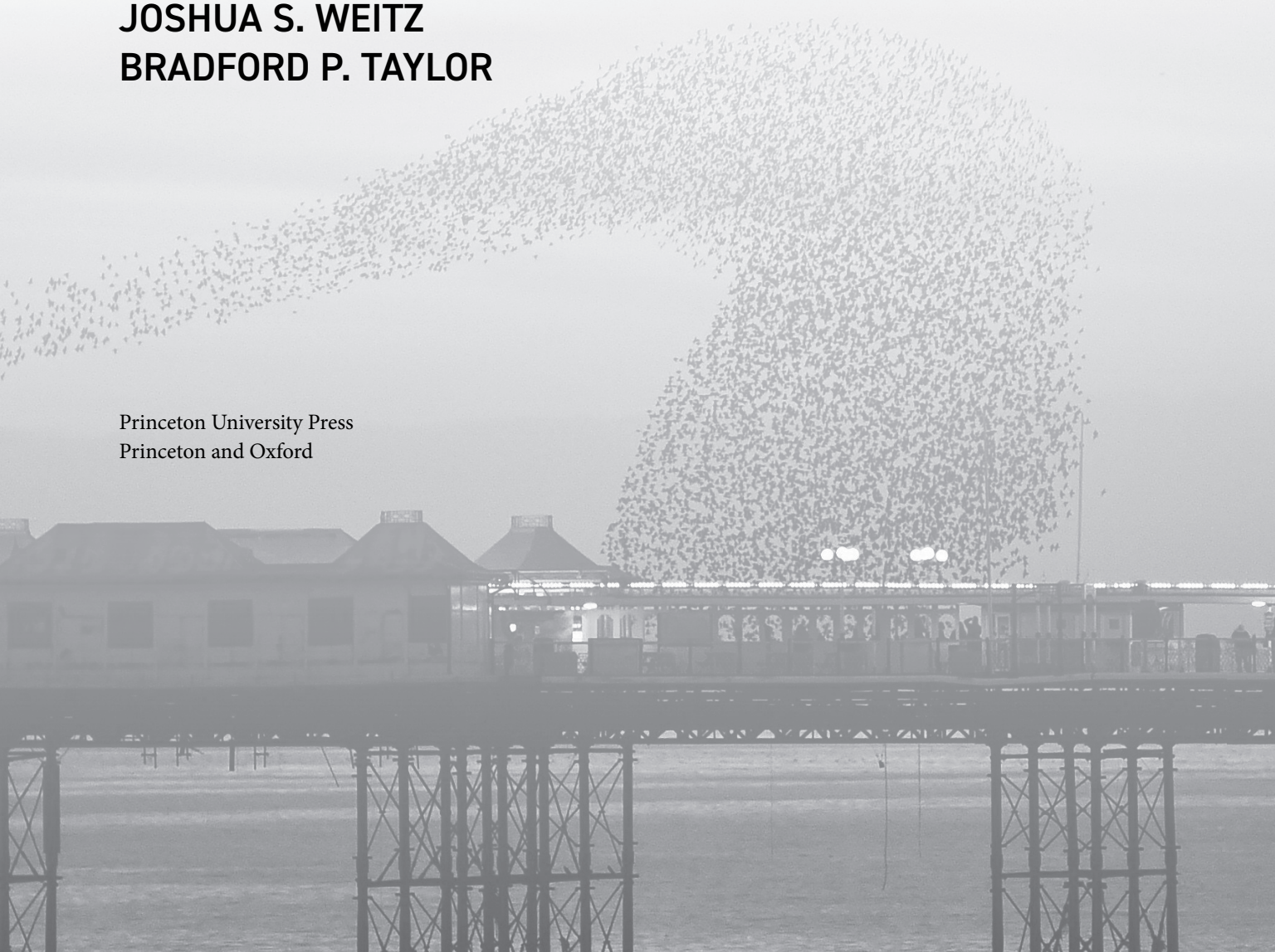
Dynamics across Cells, Organisms, and Populations

COMPANION IN MATLAB

QUANTITATIVE BIOSCIENCES

Dynamics across Cells, Organisms, and Populations

JOSHUA S. WEITZ
BRADFORD P. TAYLOR



Princeton University Press
Princeton and Oxford

Published by Princeton University Press
41 William Street, Princeton, New Jersey 08540
99 Banbury Road, Oxford OX2 6JX

press.princeton.edu

ISBN (pbk.) 9780691255682
ISBN (e-book) 9780691259628

Library of Congress Control Number: 2023946917

British Library Cataloging-in-Publication Data is available

Editorial: Sydney Carroll and Johannah Walkowicz
Production Editorial: Terri O'Prey
Text Design: Wanda España
Cover Design: Wanda España
Production: Jacqueline Poirier
Copyeditor: Jennifer McClain

Cover image: Simon Dack News / Alamy Stock Photo

This book has been composed in MinionPro and Omnes

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

CONTENTS

PREFACE

THE GOAL

This computational laboratory guide accompanies the textbook *Quantitative Biosciences: Dynamics across Cells, Organisms, and Populations*. The guide is written with students and early career scientists in mind. The near-term goal is simple: to translate biological principles and mathematical concepts into computational models of living systems. The use of computation is key. Developing computational models both democratizes and broadens the range of students from diverse backgrounds who can meaningfully integrate mathematical principles and biological concepts. In that sense, the long-term goal of this guide is to change the culture of how biology is taught and how biological research is conducted in practice.

As developed, the course upon which both the textbook and these lab notes are based includes a recurring structure. Each week is centered upon a focal scale and biological question. Are mutations dependent on selection or independent of selection? How do bacteria sense and respond to their environment? How do neurons and cardiac cells filter and integrate signals? How do individuals in a collective spontaneously move in flocks? How does rapid evolutionary change modify the dynamics of populations? These and other questions motivate a series of two lectures totaling approximately 3 hours that include a mix of biologically focused slides, an introduction to and derivation of mathematical concepts, and a journal club–like paper discussion. Then, on the third day, the class meets in a "laboratory" format for another three hours. Each student has their own computer. In front of them is a student version of that week's laboratory guide. There are no files to download, at least not typically. Instead, the laboratory guide includes code in MATLAB, Python, or R that engages with the themes and questions of the week. The students enter the code because it turns out that the act of typing reinforces concepts they already understand and highlights concepts or practices they do not yet understand.

For example, in the first week, we explore evidence that reveals whether mutations are dependent on or independent of selection. The textbook details both the biological evidence and mathematical concepts that underlie Luria and Delbrück's (LD) conclusion that mutations are random and independent of selection (Luria and Delbrück 1943). Yet, if students are to truly commit these concepts into their own practice, they must strive to re-create them. In a Quantitative Biosciences class, such an effort could involve asking students to (re)derive the Luria and Delbrück distribution or its moments. However, doing so would likely preclude many students who grasp the core idea underlying the LD mechanism but find rigorous mathematical derivations do not add to their intuition. Instead, we take a different tack. We ask students to translate the mathematical concepts into a computational model, and then probe the qualitative and quantitative behavior of the model in regimes

that reflect the biological system at hand. The art of building the model reinforces and deepens student intuition. This is not just a matter of convenience. For many systems of interest, e.g., neuronal firing, flocking of organisms, or eco-evolutionary dynamics, there may not exist a closed form solution to derive (or if one exists, it may reflect only a partial, asymptotic, or even approximate solution). In essence, computation is an imperative, not just an alternative.

This book embraces the imperative to create models of living systems for yet another purpose: to bridge the gap between receiving information from the instructor and reaching a deeper understanding. This bridging of the gap has been described by the eighteenth-century chemist Joseph Priestly as "something that cannot be described in words." David Kaiser (2005) elaborated on this concept in assessing the history of the dissemination of Feynman diagrams in the post-WWII era:

> Experimentalists must work hard to hone something like artisanal knowledge or craft skill in addition to an understanding of general principles. Historians and sociologists have argued that tacit knowledge plays a central role when it comes to replicating someone else's instruments, even when the would-be replicator is already an expert experimentalist or instrument maker.

Replace the word *experimentalists* with *quantitative bioscientists* and the word *instruments* with *code* and this quote animates the core of this hands-on computational laboratory guide. The tacit knowledge helps steer students toward good coding practice, clear and understandable modeling, and accessible visualizations of model results.

And yet there is one more objective of this computational guide. As taught in a course format, the computational laboratory guide is meant to help students prepare for homework that encourages independent thinking, problem solving, and ultimately independent research. To do so requires that students are prepared with a diverse repertoire of computational skill sets. This rationale is similar to that underlying the proliferation of tool-centric coding workshops in biology. As is apparent, if students cannot load, analyze, and visualize their data in a rigorous and repeatable manner, that will undermine the quality of well-designed experiments and sampling schemes. Yet analyzing data in the absence of biological questions can become a hollow enterprise. If students are to understand how feedbacks in living systems lead to emergent phenomena not necessarily embedded in the properties of individual components, then they need to develop a diverse repertoire of simulation approaches, to build the appropriate simulation model at the appropriate scale.

The goal of teaching practical skills as a means to increase tacit knowledge is embedded in each module. The modules themselves are not organized by method but by problem, in parallel to the organization of the main text. But the methods do in fact build upon each other. Each module introduces and/or reinforces different practical approaches to develop computational models of living systems. Students who work their way through this book should expect to gain practical expertise in the following methods:

- Sampling from probability distributions
- Stochastic branching processes
- Continuous time modeling

- Local stability analysis for nonlinear dynamical systems
- Stochastic modeling via the Gillespie algorithm
- Markov chains
- Bifurcation analysis
- Excitable system dynamics
- Partial differential equations
- Comparing stochastic to continuous models
- Agent-based simulations
- Discrete time dynamics—including the emergence of chaos

This is a non-trivial list. The mathematics underlying each approach could involve an entire course. But that is not the point. The point is to help students integrate each of these concepts when necessary into usable and principled code. In essence, the start of the title—*Quantitative Biosciences*—is meant to tell the story, emphasizing the essential integration of mathematical reasoning and computation in enabling understanding and eventually the ability of students to make advances of their own. This is a middle path, but one that we hope proves productive in the long term. Students interested in advanced studies in stochastic processes, nonlinear dynamics, or time series analysis are encouraged to pursue them. Perhaps they might be even more likely to do so after taking this course.

YOU CAN DO IT

After some debate and sufficient reflection, we have decided to replicate the course experience by organizing each chapter in the format of a student version. This means that chapters are structured as guides with questions, rather than manuals with solutions. Typically, the instructor version with solutions to challenge problems is handed out at the end of class. Here the answers are provided at the end of chapters. To get the most out of this book, we recommend that you try, as long as is possible, to avoid turning to these pages. Think of it as your own marshmallow test. The answers are there, but they are not the only answers. More than anything, the answers are there to ensure that when you get stuck there is a light to help keep you going along the path.

What then should you, the reader, do? Our recommendation: Just begin. This laboratory guide comes in three flavors: MATLAB, Python, and R. The descriptions and mathematics are common to all, so it is up to you to choose a language and stick with it. As applicable, source code is available at the book's website, free for download, use, and reuse. In addition, an optional tutorial in each language is also available on the website, and may represent a necessary primer for some students before beginning. Many other tutorials exist—but a lack of deep experience in coding should not prevent you from beginning. Why? Because you can do it.

ACKNOWLEDGMENTS

The central goal of this computational guide is to help students develop the tacit knowledge needed to explore living systems across scales. As noted in the accompanying textbook, this guide would not have been possible without the input of colleagues, students in the Quantitative Biosciences (QBioS) PhD program at Georgia Tech, and those who utilized material under development as part of undergraduate and graduate programs and as part of short courses in the United States, France, Italy, and Brazil. Special thanks go to QBioS and Georgia Tech students who provided critical feedback that has shaped the need for the book and the nature of the book itself: Qi An, Akash Arani, Emma Bingham, Pablo Bravo, Alfie Brownless, Rachel Calder, Alexandra Carruthers-Ferrero, Hyoann Choi, Ashley Coenen, Shlomi Cohen, Raymond Copeland, Sayantan Datta, Kelimar Diaz, Robert Edmiston, Shuheng Gan, Namyi Ha, Hayley Hassler, Maryam Hejri Bidgoli, Lynn (Haitian) Jin, Elma Kajtaz, Cedric Kamalseon, Katalina Kimball-Linares, Tucker J. Lancaster, Daniel A. Lauer, Zewei Lei, Guanlin Li, Hong Seo Lim, Ellen Liu, Lijiang Long, Katie MacGillivray, Jiyeon Maeng, Andreea Magalie, Pedro Márquez-Zacarías, Zachary Mobille, Daniel Muratore, Carlos Perez-Ruiz, Aaron R. Pfennig, Rozenn Pineau, Brandon Pratt, Joy Putney, Aradhya Rajanala, Athulya Ram, Elisa Rheaume, Rogelio A. Rodriguez-Gonzalez, Benjamin Seleb, Varun Sharma, Benjamin Shipley, Cassie Shriver, Michael Southard, Sarah Sundius, Disheng Tang, Stephen Thomas, Kai Tong, Akash Vardhan, Hector Augusto Velasco-Perez, Ethan Wold, Fiona Wood, Leo Wood, Siya Xie, Christopher Zhang, Mengshi Zhang, Conan Y. Zhao, and Baxi Zhong.

Thank you to Van Savage, David Murrugarra, Rafael Peña Miller, and Carles Tardío Pi for their comprehensive review of the textbook and laboratory guides. Van gets a double thanks for his willingness to try out this material in formative stages—thanks also to Tianyun Lin for facilitating feedback from UCLA students that has been essential to refining the format and content of the book. Thank you to colleagues at Georgia Tech, especially those in the Physics of Living Systems program, for their input that shaped the computational lab guide material over many years, especially Flavio Fenton, J. C. Gumbart, Simon Sponberg, and Daniel Goldman, as well as to current and former colleagues in Biological Sciences, especially Will Ratcliff as well as multiple group members who helped teach part of the course and whose input was critical to improve the material: Stephen Beckett, David Demory, Adriana Lucia Sanz, Jeremy Harris, Joey Leung, and Jacopo Marchi. We have tried to follow the good counsel of our colleagues—any remaining mistakes are ours alone.

The time and resources to develop this computational lab guide have been made possible, in part, by grants and support from the National Science Foundation (NSF) Physics of Living Systems, Biological Oceanography Dimensions of Biodiversity, Bridging Ecology and Evolution programs, NIH NIGMS, NIH NIAID, Army Research Office, Charities

in Aid Foundation, Marier Cunningham Foundation, Chaire Blaise Pascal Program of the Île-de-France, Mathworks Corporation, Simons Foundation, Centers for Disease Control and Prevention, A. James & Alice B. Clark Foundation, and the Burroughs Wellcome Fund.

Finally, thank you to the entire staff at Princeton University Press, including Alison Kalett, Sydney Carroll, and the copyeditors, illustrators, indexers, and production specialists who have elevated this material into an integrated whole.

Part I

Molecular and Cellular Biosciences

Chapter One

Fluctuations and the Nature of Mutations

1.1 HANDS-ON APPROACH TO MUTATIONS AND SELECTION

The goal of this lab is to simulate a growing bacterial population, including the ancestral "wild type" as well as mutants generated de novo during the growth process. The core techniques are straightforward: connecting the simplest model of exponential growth with stochastic events. To do so requires a few techniques, all centered on the ramifications of sampling from random distributions using MATLAB. As you will see, learning how to sample from random distributions will be relevant in many biological systems. Indeed, being able to simulate stochastic dynamics is key for simulating biological systems at scales from molecules to organisms to ecosystems. Hence, this opening chapter introduces basic concepts that are used throughout the laboratory guide. This chapter also serves another function: to link the material in the textbook with the homework.

The laboratory will prepare you to build components of two categories of mutational models, as illustrated in a generalized schematic form in Figure 1.1. These initial components form the basis for the homework problems presented in the main text. In this figure, the left panel illustrates a branching process in which an individual bacterium in generation $g = 0$ divides so that there are two bacteria in generation $g = 1$, four bacteria in generation $g = 2$, and so on such that there are 2^g bacteria after g generations. Of these, a fraction of the offspring may be different than the ancestral wild type. These different bacteria are referred to as *mutants*. Notably, in this model, mutants give rise to mutant daughter cells and not to wild-type cells. The right panel illustrates an alternative model of mutation, in which many bacteria in a single generation undergo some stochastic change, i.e., a mutation, rendering a small number of bacteria into mutants. This latter case may be related to a phenotypic change, e.g., exposure to a virus or chemical agent. How to build models of both kinds, how to compare them, and how to reconcile the predictions of such models with experimental data from Luria and Delbrück form the core of this laboratory.

The key aim of this laboratory is to begin a process to relate the mechanism by which mutants are generated with signatures that can be measured. These signatures may include the mean as well as the variance in the number of mutants between parallel experiments.

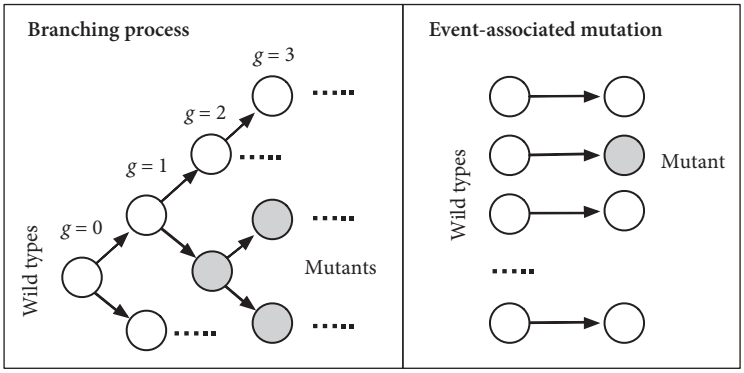


Figure 1.1: Stochastic models of mutation: Mutations are independent of selection (left) or dependent on selection (right). (Left) Branching process in which a single (or small) number of wild-type bacteria (empty cells) divide and occasionally mutate; the mutants (shaded) also divide. (Right) Mutation occurs randomly among a large population given interaction with a selective pressure, leading to a small fraction of mutants.

1.2 SAMPLING FROM PROVIDED DISTRIBUTIONS

In order to simulate stochastic processes, such as mutation in a population, one must repeatedly sample random numbers. Random numbers can be generated by any modern programming language. In doing so, it is possible to use built-in functions or to manipulate the generated random numbers to ensure they have a specified mean, variance, and higher-order moments. For example, to randomly sample a number between 0 and 1, use the command

```
rand
```

Do this a few times. Each number is different. But generating multiple random numbers one at a time is unnecessary. Instead, generating multiple random numbers can be done automatically; e.g., use the following commands to randomly sample 100 points between 0 and 1:

```
randvec = rand(1,100);
```

or

```
randvec = rand(100,1);
```

These commands will generate a set of 100 random numbers either in a row, or a column.

It is also possible, as shown in the introductory coding demos available on the book's website, to generate random matrices. Use the following command to generate a random matrix of size $m \times n$ array:

```
randarray = rand(m,n);
```

As is apparent, the shape of the matrix can be specified in terms of the number of rows `m` and columns `n`. If the code does not work, that is probably because you have not yet defined

the size; do so a few times and see how easy it is to generate distinct random matrices. Note for future reference that two arrays must be the same size in order to perform element-wise operations (e.g., addition, subtraction, or element-by-element multiplication). Also note the names of variables—they tend to be descriptive. This is a good practice because it makes code easier to read, modify, and reuse. The first challenge problem should help get you more comfortable working with the core features of random distributions.

CHALLENGE PROBLEM: Properties of Random Distributions

What is the mean value of a single instance of invoking `rand`? Similarly, what is the variance? Once you have identified the mean and variance, plot the distribution of numbers generated by `rand` by sampling a large number of points (10^4) and then using the `hist` function to generate a histogram. What shape is the distribution? How does it change as you change the number of bins for the histogram?

MATLAB also allows sampling different distributions than the uniform distribution. As one exercise, plot the distribution of the output for the following functions: (i) standard normal distribution with a mean of 20 and standard deviation of 5 using

```
randn
```

and (ii) the Poisson distribution with rate parameter $\lambda = 20$ using

```
poissrnd
```

Examples of the outputs can be seen in Figure 1.2.

It is possible to shift the range of randomly generated numbers using relatively simple operations, generating arbitrary variations (in range and location) of preexisting distributions. This may be useful in many circumstances, not only within the context of the LD

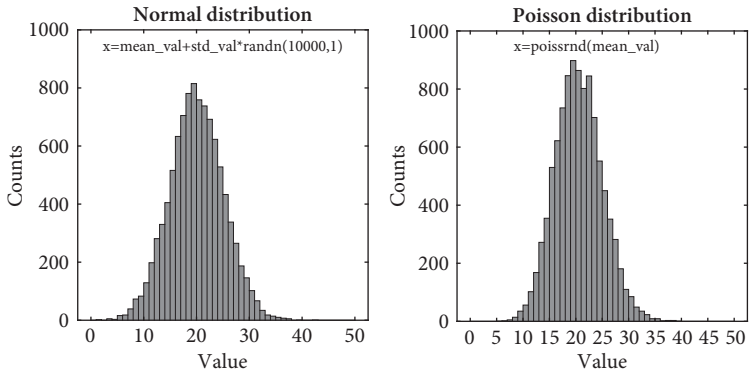


Figure 1.2: Sampling from random distributions, including the normal distribution (left) and the Poisson distribution (right).

problem. The following challenge problem provides an opportunity to build your intuition for manipulating and generating random numbers with distinct means and ranges.

CHALLENGE PROBLEM: Random Number Generation

This problem focuses on modifying the means and ranges of random numbers by modulating the output of built-in random number functions.

- Generate 1000 random numbers equally spaced between 0 and 5.
- Generate 1000 random numbers equally spaced between 2 and 7.
- Generate 1000 random numbers equally spaced between −5 and 5.

In each of these cases, use the built-in random number generator and then simple arithmetic (i.e., addition, subtraction, and multiplication) to transform the random numbers to specified ranges. You can do it!

1.3 SAMPLING FROM CUSTOM DISTRIBUTIONS

MATLAB offers the option to generate specialized distributions. However, it is also possible to sample from "custom" distributions, i.e., both parametric and non-parametric distributions. One way to do so is to leverage the *cumulative distribution function*, or cdf. The cdf at a point, x, gives the probability of observing a value less than or equal to x. Formally, if $p(x)\mathrm{d}x$ is the probability of observing the random variable between x and $x + \mathrm{d}x$, then the cdf is

$$P(x) = \int_{-\infty}^{x} p(y)\mathrm{d}y \tag{1.1}$$

where y is a "dummy" variable used here for notational purposes of integrating over the probability distribution. The cdf is a monotonically increasing function with a range between 0 and 1. These constraints allow random sampling from arbitrary distributions if one is provided with the cdf in advance, by leveraging properties of the uniform distribution. An ideal way to illustrate this is via the exponential distribution.

The exponential distribution arises in many biological processes. For example, for processes that randomly occur with a constant rate λ, then the time of the first occurrence of an event is exponentially distributed such that $p(x) = \lambda e^{-\lambda x}$, given mean time $1/\lambda$. The cdf of the exponential distribution is $1 - e^{-\lambda x}$. Most numerical software tools have packages to sample exponential random numbers; this is precisely why it is instructive to compare the built-in solution to the custom solution. Indeed, one can think of the cdf of the exponential random distribution as having a one-to-one correspondence with the cdf of the uniform random distribution. That is, whereas half the values generated by a uniform random distribution will be < 0.5, that is not true for an exponential distribution. Instead, given the shape parameter λ, then half the values of an exponential distribution will have values $x < x_u$ such that $1 - e^{-\lambda x_u} = 0.5$. This insight can help move from one distribution to the other.

To sample random numbers from the exponential distribution, first sample from the uniform distribution between 0 and 1.

```
probsamp = rand;
```

Think of this as a random value of P, which we denote as c_u. By randomly sampling the cdf of the uniform random distribution, the next question becomes: what value of the exponentially distributed random variable x_e corresponds to that point in the cdf? To answer this question requires that we invert the cdf, i.e., $P = 1 - e^{-\lambda x_e}$, to obtain an equation of x_e in terms of the cdf. To show this in action, denote c_u as the randomly selected value from the cdf of the uniform distribution. To map the cdf of the uniform distribution onto the cdf of the exponential distribution (our custom distribution) requires that $c_u = 1 - e^{-\lambda x_e}$. For x_e to be an exponentially distributed random number requires transforming the uniform random numbers into the exponentially distributed random numbers we would like to generate:

$$\begin{aligned} 1 - e^{-\lambda x_e} &= c_u \\ e^{-\lambda x_e} &= 1 - c_u \\ -\lambda x_e &= \log 1 - c_u \\ x_e &= \frac{-\log 1 - c_u}{\lambda} \end{aligned} \tag{1.2}$$

This gives $x_e = -\frac{1}{\lambda}\log(1 - c_u)$. This converts the sampling distribution of `rand` (i.e., c_u) to an exponential distribution. In order to use this repeatedly, it will be convenient to make and save a function:

```
function exprand = rand2exp(probsamp,lambda)
  exprand = -1/lambda*log(1-probsamp);
end
```

The following section leverages the prior code snippet for using uniform sampling to generate exponentially distributed numbers given a process with rate $\lambda = 1/10$:

```
lambda=1/10;
probsamp = rand(10^3,1);
expprobsamp = rand2exp(probsamp,lambda)
```

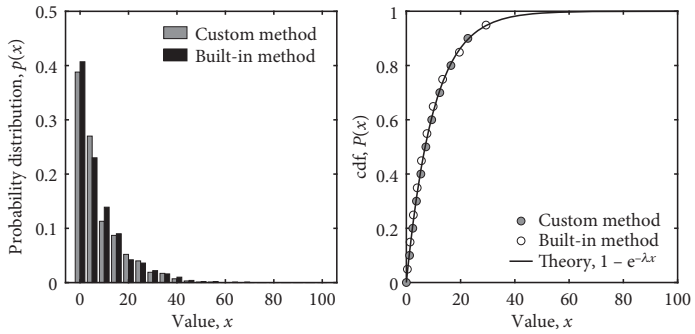
Now it is time to see if any of this works—via a challenge problem.

CHALLENGE PROBLEM: Comparing Exponential Random Sampling

Compare the distribution of 10^3 exponentially distributed random numbers using the cdf-based method to the distribution using the following built-in MATLAB command:

```
matlabexprnd = random('exp',1/lambda,10^3,1);
```

Note that the function `random` allows sampling from a number of different common distributions. (Hint: Another helpful function is the *empirical cumulative distribution function*, or `ecdf`. This is useful in generating cumulative distributions from randomly sampled "data.") If your code is working, it should look like the following:



These figures show a comparison of customized sampling and built-in exponential random sampling via probability distributions (left) and cumulative distributions (right). For the cdf, the expected distribution is shown as a solid black line.

1.4 COMPARING BINOMIAL AND POISSON DISTRIBUTIONS

Binomial distributions result from counting the number of occurrences given independent samples with probability of occurrence p. For example, consider a mutation probability of $p = 10^{-8}$. Irrespective of whether mutations are independent of or dependent on selection, it would typically take a large number of cell divisions (or cells) for a mutant to appear. Using a binomial distribution, one could, in theory, predict the number of mutants expected to occur in a single round of cell division or given an exposure of a large collection of n cells to a selective force. Formally, the binomial distribution denotes the probability that k events occur out of n trials given the per trial probability p. This distribution is

$$P(k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{1.3}$$

where $\binom{n}{k}$ denotes the number of unique ways of choosing k of n elements (i.e., the binomial coefficient). However, if occurrences are rare and the number of samples, n, is large, then the binomial distribution converges to the Poisson distribution with shape parameter $\lambda = np$ (this shape is the expected mean number of events in n trials). To see this computationally, compare the cdfs of the sample of repeated binomial sampling to repeated Poisson sampling with varying n. For example, use the following code to obtain and plot the cdf for binomial random numbers given 100 trials each with probability $p = 0.2$:

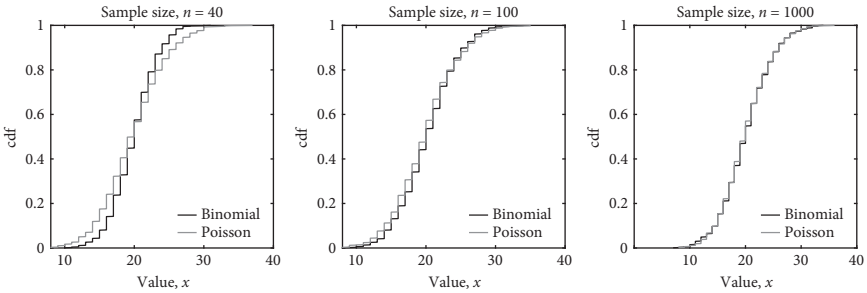
```
n = 100;
lambda = 20;
p = lambda/n;
numsamps = 10^3;
```

```
binosamps = binornd(n,p,1,numsamps);
sortbino=sort(binosamps,'ascend');
cdfbino=(1:numsamps)/numsamps;
tmph=plot(sortbino,cdfbino,'k-');
set(tmph,'linewidth',2);
```

Here `binornd` samples binomial random numbers, and sorting the result allows for an explicit calculation of the empirical cdf (without using a built-in function).

CHALLENGE PROBLEM: Comparing the Binomial to the Poisson

Compare the binomial and Poisson cdfs for $n = 40$, 100, and 1000 in each case, assuming there is an expected number of 20 events such that the probability per event decreases from 0.5 to 0.2 to 0.02, respectively. If your code is working, it should look something like this:



Technical note: Keep in mind that the `binornd` function generates the outcome of `n` trials each with a `p` probability of success. Try to compare outcomes with `sum(rand(n,1)<p)`. Are the outcomes different in a substantive way than simply sampling from a binomial? It is worth considering that the binomial distribution is equivalent to running n trials each with a p probability of success and then reporting how many, m, were successful. By definition, $0 \leq m \leq n$. Hence, each trial is successful with probability p. Because `rand` returns uniformly distributed random numbers between 0 and 1, then `rand < p` is 1 with probability p and 0 with probability $1 - p$. As such, by invoking the `rand` command n times and comparing it to p, it should return a 1 approximately np times; this is, by definition, λ from above. Hence, just as we used the uniform random distribution to generate exponentially distributed numbers, it is also possible to use the same distribution to generate random events that have precisely the same properties as binomial random numbers.

1.5 THE START OF DYNAMICS

The schematics in Figure 1.1 illustrate two distinct mechanisms by which mutant bacteria can increase in number in a population. Via the independent mutation hypothesis, mutations happen rarely during cell division and then selection acts upon them later. Via

the dependent mutation hypothesis, mutations only occur when the cell experiences a selection pressure, and in that case a small fraction of heritable cells acquire a mutation. The consequences of these two ideas are examined at length in the main text and then developed as the centerpiece of the homework problems. Yet to get there requires that you develop a dynamic simulation.

Rather than giving away the homework (and the fun involved in doing this yourself), there is a way to start along the path toward dynamics. First, consider the case where mutations are dependent on selection. It should be apparent that manipulating probability distributions as described here can be used to generate a small number of mutants in a population. For example, consider the case where there are $n = 10^5$ cells and $p = 10^{-4}$. In that event, one expects approximately 10 mutational events, which can be generated as follows: `sum(rand(n,1)<p)`. Yet the case of the independent selection is more difficult.

As a start to a dynamic model, consider a two-step process. First, a population of cells, with certain features doubles in size. Second, a fraction of the cells changes in some way. Simply to illustrate this point, initialize a 1× 5 array with 0.5 in the second entry, e.g.,

```
x=zeros(1,5);
x(2)=0.5;
```

Next, double the size of the array. How to do this is up to you. Indeed, doubling an array is perhaps a crude way to simulate a dynamic process, but it provides some intuition to the underlying changes in the system. It also helps illustrate ways to concatenate matrices together, e.g., `y=[x,x];` Examine the output of `y` and notice that there are now instances of a 0.5 value, in the second and seventh positions. This is a direct result of concatenating the matrices. Now, if the value of 0.5 was some property of a cell, then it is clear that two cells have that same property. If instead one used a value of 1 for a mutant and 0 for a wild type, then it is apparent that the process of cell division (which doubled the number of cells) also doubled the number of mutants. Of course, at this point it would be important to change the property of `y` in the event of a new mutation. In that case, you can use the random number generating methods already described to decide which, if any, of the array elements to change.

Of course, if you want to see what happens in a few instances, consider this loop:

```
x=[1,0];
for i=1:4,
  x=[x,x]
end
```

The result should be a growing list of 0s and 1s:

```
1  0
1  0  1  0
1  0  1  0  1  0  1  0
1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0
1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 ...
```

This kind of approach loses track of the mother-daughter relationships (at least explicitly). But it is possible to modify the arrays and then begin to change both the size and the nature of the population.

How to build models of bacterial growth and mutation is treated in detail in the textbook (and associated homework). From a computational perspective, such models are built around a few simple ideas, including adding elements to an array and changing the value of an array. For example, here are a series of small exercises that illustrate core concepts toward building your own simulation model of bacterial growth and mutation. Type in each and modify them. Soon you may just be ready to tackle the question of whether mutations are dependent on or independent of selection.

CHALLENGE PROBLEM: A Step toward Bacterial Growth

Write a program to generate an in silico population of 100 bacteria, of which ≈ 90% are wild types and the rest are mutants (denote these as 1s and 0s, respectively). Then double the size of this population while retaining the properties of the original population. Finally, switch one element, either 0 to 1 or 1 to 0.

1.6 INFERRING PARAMETERS FROM DATA

Thus far, this laboratory has provided resources for sampling from and manipulating different probability distributions—with an eye toward developing dynamic simulations of growing and mutating bacterial populations. These can be used in a generative sense, as described in the textbook, to compare and contrast the independent and acquired mutational hypotheses. However, there is another question that is relevant to hypothesis testing: how to infer process rates and parameters from data. To tackle such an approach, first download the file `poissdata.csv`, which contains 100 random samples from Poisson distributions with an unknown rate parameter. Or you can enter the following string of numbers into an array. Here it is—exciting, no?

3,4,2,5,2,2,5,0,5,2,4,4,4,1,4,3,3,2,3,2,2,6,3,4,4,5,2,2,5,0,1,2,2,2,4,3,
3,2,4,5,2,4,6,3,5,5,1,3,1,2,2,5,4,8,4,3,5,2,6,3,3,2,3,4,4,3,2,2,3,2,6,2,
2,0,2,5,4,5,4,5,3,9,3,5,2,6,3,5,1,1,2,1,4,2,5,7,4,3,4,4

Although this seems abstract, imagine that these numbers correspond to resistance colonies measured after a Luria-Delbrück (LD) experiment—it turns out that these have features quite distinct from the LD experiments, but they nonetheless provide a good basis for deeper exploration. The remainder of this lab is aimed at estimating the rate parameter, i.e., the unknown λ, from which one could estimate the unknown mutation rate. These steps are the centerpiece of the homework. Hence, it's worthwhile to take some time to understand the *inverse* problem using a simpler example.

The central objective of parameter inference is to try to identify a value (or set of values) that is compatible with observations. The degree of compatibility may depend on one's preference for the unexpected. In practice, most inference approaches try to ask the