Dilranjan S. Wickramasuriya
Rose T. Faghih

Bayesian Filter Design for Computational Medicine

A State-Space Estimation Framework

Springer

Bayesian Filter Design for Computational Medicine

Dilranjan S. Wickramasuriya • Rose T. Faghih

Bayesian Filter Design for Computational Medicine

A State-Space Estimation Framework

Springer

Dilranjan S. Wickramasuriya
hSenid Mobile Solutions
Colombo, Sri Lanka

Rose T. Faghih
Biomedical Engineering Department
New York University
New York, NY, USA

ISBN 978-3-031-47103-2 ISBN 978-3-031-47104-9 (eBook)
https://doi.org/10.1007/978-3-031-47104-9

This work was supported by New York University.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Paper in this product is recyclable.

In honor of Professor Emery N. Brown's belated 65th birthday
In honor of Professor Munther Dahleh's belated 60th birthday
In honor of Professor Sridevi Sarma's belated 50th birthday
In honor of Professor George Verghese's 70th birthday

Preface

Point processes underlie a range of activities within the human body. Neural spiking, rhythmic cardiac contraction, and pulsatile hormone secretion all have binary-like mechanisms at their core. The field of estimating latent states tied to point process observations has seen a steady growth over a period that has now exceeded two decades. These state estimation methods have found applications across a range of specialities including behavioral learning, brain-computer interfaces, sleep studies, heart rate variability analysis, anesthesia, endocrinology, and human emotion. The field has also seen an expansion from some of the early state-space estimators that were limited to point process observations alone to mixed estimators that can incorporate both binary and continuous-valued observations.

Despite the growth in the field and its widespread applicability, several challenges are encountered by those with an undergraduate engineering degree who wish to begin developing these types of estimators. While the estimators are similar to regular Kalman filters, their design is not typically approached in the way that regular Kalman filters are. Instead, the design of state estimators for point process observations is usually approached from a statistical Bayesian point-of-view, rather than from the typical least squares minimization perspective. Moreover, while a number of works can be found in the literature involving these types of point process Bayesian filters, there is no tutorial-like introduction to aid the beginner. Consequently, the student who wishes to begin research has to spend considerable time to learn the basics of point process Bayesian filter design; and that often from material in research papers which are not intended as tutorials. This book is an attempt to bridge the gap. Hence, an intended reader wishing to learn filter design is expected to have taken an undergraduate course in basic probability and statistics as well as a course in signals and systems. Some background in computer programming would also be necessary to implement the filters. A course in control systems, although not required, would be helpful as well. A reader merely intending to use the filters presented here, however, would not need this background but only require some basic proficiency with MATLAB.

Point process state estimators have found a number of applications in fields related to physiology and medicine, some of which have been listed above. This

is partly due to the prevalence of point processes phenomena in different types of physiological signals. Successful collaborative research has also resulted based on drawing connections between state-space estimation and physiology—connections that serve to bridge the gap between engineering and medicine. Our book is also intended to serve the non-engineering community. Thus, the practitioner who does not wish to delve into all the mathematical detail underlying filter design but merely wishes to apply the tools can also use the book. Precisely for this purpose, the book is accompanied by a MATLAB toolbox of code examples that cover the different filters. Brief descriptions of the code are also provided at the end of the main chapters. It is our expectation that the endocrinologist, psychologist, or other researcher who wishes to estimate latent physiological states underlying binary and continuous-valued measurements would thus benefit as well.

Point processes are everywhere, if you look for them. Earthquakes, crime incidents, rainfall, disease infections, customer arrivals at a bank, and website visits can all be modeled using point processes. Therefore, the applicability of this book isn't solely limited to physiology. Researchers in agriculture, epidemiology, climatology, etc. whose work involves point process phenomena can all find the book a helpful aid.

The ever-increasing role of technology in our lives is undeniable. We rely on technology from everything to flying across continents to buying candy from a vending machine. It is not unlikely that the foreseeable future will involve an explosion of smart electronics everywhere. These networked devices will be connected to the cloud where more powerful analytical tools sift through the raw data. As with the case of latent variable estimation using state-space models, the merging between physical components and cyber analytics will seek to extract underlying or hidden patterns and information from the sensed data. The same scenario is also applicable to the human body. It is likely that the future will involve an increased adoption of bioelectric and biochemical sensors that will play a crucial role in our well-being. The sensed physiological signals will inevitably contain some point process data, all of which capture information regarding latent states within the human body and brain. Thus, with the aid of appropriate mathematical tools, some of which are covered in this book, a sensor-laden wristwatch in the future may be able to precisely tell you some of what's happening inside your brain and your body—an idea that led to an Innovators Under 35 recognition by MIT Technology Review for one of the authors.

It is our hope that this book will be of help to students, researchers, and practitioners alike.

Colombo, Sri Lanka — Dilranjan S. Wickramasuriya
New York, NY, USA — Rose T. Faghih

Acknowledgments

The authors gratefully acknowledge the National Science Foundation (NSF) in the writing of this book. In particular, the work was supported by the following NSF grants: 2226123/1942585—CAREER: MINDWATCH: Multimodal Intelligent Noninvasive brain state Decoder for Wearable AdapTive Closed-loop arcHitectures and 1755780—CRII: CPS: Wearable-Machine Interface Architectures. The open-access publication of this book was made possible by the support of the New York University Faculty Startup Fund.

Contents

List of Figures

Chapter 1
Introduction

The human body is an intricate network of multiple functioning sub-systems. Many unobserved processes quietly keep running within the body even while we remain largely unconscious of them. For decades, scientists have sought to understand how different physiological systems work and how they can be mathematically modeled. Mathematical models of biological systems provide key scientific insights and also help guide the development of technologies for treating disorders when proper functioning no longer occurs. One of the challenges encountered with physiological systems is that, in a number of instances, the quantities we are interested in are difficult to observe directly or remain completely inaccessible. This could be either because they are located deep within the body or simply because they are more abstract (e.g., emotion). Consider the heart, for instance. The left ventricle pumps out blood through the aorta to the rest of the body. Blood pressure inside the aorta (known as central aortic pressure) has been considered a useful predictor of the future risk of developing cardiovascular disease, perhaps even more useful than the conventional blood pressure measurements taken from the upper arm [1]. However, measuring blood pressure inside the aorta is difficult. Consequently, researchers have had to rely on developing mathematical models with which to estimate central aortic pressure using other peripheral measurements (e.g., [2]). The same could be said regarding the recovery of CRH (corticotropin-releasing hormone) secretion timings within the hypothalamus—a largely inaccessible structure deep within the brain—using cortisol measurements in the blood based on mathematical relationships [3]. Emotions could also be placed in this same category. They are difficult to measure because of their inherently abstract nature. Emotions, however, do cause changes in heart rate, sweating, and blood pressure that can be measured and with which someone's feelings can be estimated. What we have described so far, in a sense, captures the big picture underlying this book. We have physiological quantities that are difficult to observe directly, we have measurements that are easier to acquire, and we have the ability to build mathematical models to estimate those inaccessible quantities.

D. S. Wickramasuriya, R. T. Faghih, *Bayesian Filter Design for Computational Medicine*, https://doi.org/10.1007/978-3-031-47104-9_1

Let us now consider some examples where the quantities we are interested in are rather abstract. Consider a situation where new employees at an organization are being taught a new task to be performed at a computer. Let us assume that each employee has a cognitive "task learning" state. Suppose also that the training sessions are accompanied by short quizzes at the end of each section. If we were to record how the employees performed (e.g., how many answers they got correct and how much time they took), could we somehow *determine* this cognitive learning state, and see how it gradually changes over time? The answer indeed is yes, with the help of a mathematical model, we can estimate such a state and track an employee's progress over time. We will, however, first need to build such a model that relates learning to quiz performance. As you can see, the basic idea of building models that relate difficult-to-access quantities to measurements that we can acquire more easily and then estimate those quantities is a powerful concept. In this book, we will see how *state-space models* can be used to relate physiological/behavioral variables to experimental measurements.

State-space modeling is a mature field within controls engineering. In this book, we will address a specific subset of state-space models. Namely, we will consider a class of models where all or part of the observations are binary. You may wonder why binary observations are so important? In reality, a number of phenomena within the human body are binary in nature. For instance, the millions of neurons within our bodies function in a binary-like manner. When these neurons receive inputs, they either fire or they do not. The pumping action of the heart can also be seen as a binary mechanism. The heart is either in contraction and pumping out blood or it is not. The secretion of a number of pulsatile hormones can also be viewed in a similar manner. The glands responsible for pulsatile secretion are either secreting the hormone or not. In reality, a number of other binary phenomena exist and are often encountered in biomedical applications. Consequently, physiological state-space models involving binary-valued observations have found extensive applications across a number of fields including behavioral learning [4–9], position, and movement decoding based on neural spiking observations [10–17], anesthesia, and comatose state regulation [18–20], sleep studies [21], heart rate analysis [22, 23], and cognitive flexibility [9, 24]. In this book, we will see how some of these models can be built and how they can be used to estimate unobserved states of interest.

1.1 Physiology, State-Space Models, and Estimation

As we have just stated, many things happen inside the human body, even while we are largely unaware that they are occurring. Energy continues to be produced through the actions of hormones and biochemicals, changes in emotion occur within the brain, and mental concentration varies throughout the day depending on the task at hand. Despite the fact that they cannot be observed, these internal processes do give rise to changes in different physiological phenomena that can

indeed be measured. For instance, while energy production cannot be observed directly, we can indeed measure the hormone concentrations in the blood that affect the production mechanisms. Similarly, we can also measure physiological changes that emotions cause (e.g., changes in heart rate). Concentration or cognitive load also cannot be observed, but we can measure how quickly someone is getting their work done and how accurately they are performing. Let us now consider how these state-space models relate unobserved quantities to observed measurements.

Think of any control system such as a spring–mass–damper system or RLC circuit (Fig. 1.1). Typically, in such a system, we have several internal state variables and some sensor measurements. Not all the states can be observed directly. However, sensor readings can and do provide some information about them. By deriving mathematical relationships between the sensor readings and the internal states, we can develop tools that enable us to estimate the unobserved states over time. For instance, we may not be able to directly measure all the voltages and currents in a circuit, but we can use Kirchoff's laws to derive relationships between what we cannot observe and what we do measure. Similarly, we may not be able to measure all the positions, velocities, or accelerations within a mechanical system, but we can derive similar relationships using Newton's laws. Thus, a typical engineering system can be characterized via a state-space formulation as shown below (for the time-being, we will ignore any noise terms and non-linearities).

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k \tag{1.1}$$

$$\mathbf{y}_k = C\mathbf{x}_k. \tag{1.2}$$

Here, $\mathbf{x}_k$ is a vector representing the internal states of the system, $\mathbf{y}_k$ is a vector representing the sensor measurements, $\mathbf{u}_k$ is an external input, and A, B, and C are matrices. The state evolves with time following the mathematical relationship in (1.1). While we may be unable to observe $\mathbf{x}_k$ directly, we do have the sensor readings $\mathbf{y}_k$ that are related to it. The question is, can we now apply this formulation to the *human body*? In this case, $\mathbf{x}_k$ could be any of the unobserved quantities we

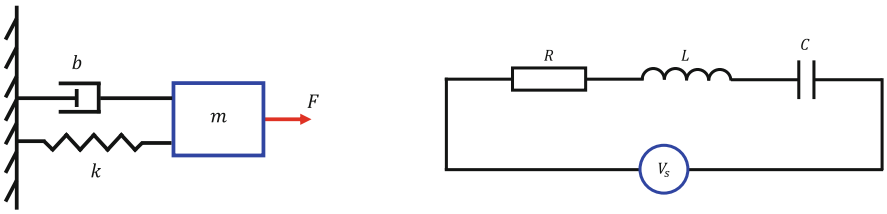

Fig. 1.1 Some examples of engineering systems that can be modeled using state-space representations. The left sub-figure depicts a spring–mass–damper system, and the right sub-figure depicts an RLC circuit. We may not be able to directly observe all the states within each system, but we can build state-space models and use whatever measurements we have to estimate them

just mentioned (e.g., energy production, emotion, or concentration) and $\mathbf{y}_k$ could be any related physiological measurement(s).

In this book, we will make use of an approach known as expectation–maximization (EM) for estimating unobserved quantities using state-space models. In a very simple way, here is what the EM algorithm does when applied to state estimation. Look back at (1.1) and (1.2). Now assume that this formulation governs how emotional states ($\mathbf{x}_k$) vary within the brain and how they give rise to changes in heart rate and sweat secretions ($\mathbf{y}_k$) that can be measured. We do not know $\mathbf{x}_k$ for $k = 1, 2, \ldots, K$, and neither do we know A, B, or C. We only have the recorded sensor measurements (features) $\mathbf{y}_k$. First, we will assume some values for A, B, and C, i.e., we will begin by assuming that we know them. We will use this knowledge of A, B, and C to estimate $\mathbf{x}_k$ for $k = 1, 2, \ldots, K$. We now know $\mathbf{x}_k$ at every point in time. We will then use these $\mathbf{x}_k$'s to come up with an estimate for A, B, and C. We will then use those *new* values of A, B, and C to calculate an even better estimate for $\mathbf{x}_k$. The newest $\mathbf{x}_k$ will again be used to determine an even better A, B, and C. We will repeat these steps in turn until there is hardly any change in $\mathbf{x}_k$, A, B, or C. Our EM algorithm is said to have *converged* at this point. The step where $\mathbf{x}_k$ is estimated is known as the *expectation*-step or E-step and the step where A, B, and C are calculated is known as the *maximization*-step or M-step. For the purpose of this book, we will label the E-step as the *state estimation step* and the M-step as the *parameter estimation step*. What follows next is a basic description of what we do at these steps in slightly more detail.

1.1.1 State Estimation Step

As we have just stated, our EM algorithm consists of two steps: the state estimation step and the parameter estimation step. At the state estimation step we assume to know A, B, and C and try to estimate $\mathbf{x}_k$ for $k = 1, 2, \ldots, K$. We do this *sequentially*. Again, look back at (1.1) and (1.2). Suppose you are at time index k and you know what A, B, C, and $\mathbf{x}_{k-1}$ are, could you come up with a guess for $\mathbf{x}_k$? You can also assume that you know what the external input $\mathbf{u}_k$ is for $k = 1, 2, \ldots, K$. How would you do determine $\mathbf{x}_k$? First, note that we can re-write the equations as

$$\mathbf{x}_k = A\mathbf{x}_{k-1} + B\mathbf{u}_{k-1} \tag{1.3}$$

$$\mathbf{y}_k = C\mathbf{x}_k. \tag{1.4}$$

If you knew A, B, C, $\mathbf{x}_{k-1}$, and $\mathbf{u}_{k-1}$, and had to determine $\mathbf{x}_k$ just at time index k, you would encounter a small problem here. Do you see that $\mathbf{x}_k$ appears in both equations? You could simply plug-in the values of $\mathbf{x}_{k-1}$ and $\mathbf{u}_{k-1}$ into (1.3) and get a value for $\mathbf{x}_k$. Since you are using the *past* values up to time index $(k-1)$ to determine $\mathbf{x}_k$, this could be called the *predict* step. You are done, right? Not quite. If

you determine $\mathbf{x}_k$ solely based on (1.3), you would *always* be discounting the sensor measurement $\mathbf{y}_k$ in (1.4). This sensor measurement is also an important source of information about $\mathbf{x}_k$. Therefore, at each time index k, we will first have the predict step where we make use of (1.3) to guess what $\mathbf{x}_k$ is, and then apply an *update* step, where we will make use of $\mathbf{y}_k$ to improve the $\mathbf{x}_k$ value that we just predicted. The full state estimation step will therefore consist of a series of repeated predict, update, predict, update, ... steps for $k = 1, 2, \ldots, K$. At the end of the state estimation step, we will have a complete set of values for $\mathbf{x}_k$.

Dealing with uncertainty is a reality with any engineering system model. These uncertainties arise due to noise in our sensor measurements, models that are unable to fully account for actual physical systems and so on. We need to deal with this notion of uncertainty when designing state estimators. To do so, we will need some basic concepts in probability and statistics. What we have said so far regarding estimating $\mathbf{x}_k$ can be mathematically formulated in terms of two fundamental ideas in statistics: mean and variance. In reality, (1.3) and (1.4) should be

$$\mathbf{x}_k = A\mathbf{x}_{k-1} + B\mathbf{u}_{k-1} + \mathbf{e}_k \tag{1.5}$$

$$\mathbf{y}_k = C\mathbf{x}_k + \mathbf{v}_k, \tag{1.6}$$

where $\mathbf{e}_k$ is what we refer to as *process noise* and $\mathbf{v}_k$ is *sensor noise*. Therefore, when we "guess" what $\mathbf{x}_k$ is at the predict step, what we are really doing is determining the *mean* value of $\mathbf{x}_k$ given that we have observed all the data up to time index $(k-1)$. There will also be a certain amount of uncertainty regarding this prediction for $\mathbf{x}_k$. We quantify this uncertainty in terms of *variance*. Thus we need to determine the mean and variance of $\mathbf{x}_k$ at our predict step. But what happens after we observe $\mathbf{y}_k$? Again, the idea is the same. Now that we have two sources of information regarding $\mathbf{x}_k$ (one based on the prediction from $\mathbf{x}_{k-1}$ and $\mathbf{u}_{k-1}$, and the other based on the sensor reading $\mathbf{y}_k$), we will still be determining the mean and variance of $\mathbf{x}_k$. So we need to calculate *one* mean and variance of $\mathbf{x}_k$ at the predict step, and *another* mean and variance of $\mathbf{x}_k$ at the update step.

1.1.2 Parameter Estimation Step

Recall that our EM algorithm iterates between the state estimation step and the parameter estimation step until convergence. Assume that we sequentially progressed through repeated predict, update, predict, update, ... steps for $k = 1, 2, \ldots, K$ and determined a set of mean and variance (uncertainty) values for $\mathbf{x}_k$. How could we use all of these mean and variance values to determine what A, B, and C are? Here is how we proceed. We first calculate the *joint probability* for all the $\mathbf{x}_k$ and $\mathbf{y}_k$ values. The best estimates for A, B, and C are the values that *maximize* this probability (or the log of this probability). Therefore, we need to maximize this probability *with respect to* A, B, and C. One simple way to determine the value at

which a function is maximized is to take its derivative and solve for the location where it is 0. This is basically what we do to determine A, B, and C (in reality, we actually maximize the *expected value* or *mean* of the joint log probability of all the $\mathbf{x}_k$ and $\mathbf{y}_k$ values to determine A, B, and C).

1.1.3 Algorithm Summary

In summary, we have to calculate means and variances at the state estimation step and derivatives at the parameter estimation step. We will show how these equations are derived in a number of examples in the chapters that follow. The EM approach enables us to build powerful state estimators that can determine internal physiological quantities that are only accessible through a set of sensor measurements.

What we have described so far is a very simple introduction to the EM algorithm as applied to state estimation. Moreover, for someone already familiar with state-space models, the predict and update steps we have just described should also sound familiar. These are concepts that are found in *Kalman filtering*. The derivation of the Kalman filter equations is generally approached from the point of view of solving a set of simultaneous equations when new sensor measurements keep coming in. In this book, we will not approach the design of the filters through traditional *recursive least squares minimization* approaches involving matrix computations. Instead, we will proceed from a statistical viewpoint building up from the basics of mean and variance. Nevertheless, we will use the terminology of a *filter* when deriving the state estimation step equations. For reasons that will become clearer as we proceed, we can refer to these state estimators as *Bayesian* filters.

1.2 Book Outline

State-space models have been very useful in a number of physiological applications. In this book, we consider state-space models that give rise, fully or partially, to binary observations. We will begin our discussion of how to build Bayesian filters for physiological state estimation starting with the simplest cases. We will start by considering a scalar-valued state x_k that follows the simple *random walk*

$$x_k = x_{k-1} + \varepsilon_k, \tag{1.7}$$

where $\varepsilon_k \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ is process noise. We will consider how to derive the state and parameter estimation step equations when x_k gives rise to a single binary observation n_k. We will next proceed to more complicated cases. For instance, one of the cases will be where we have a *forgetting factor* ρ such that

$$x_k = \rho x_{k-1} + \varepsilon_k, \tag{1.8}$$

and x_k gives rise to both a binary observation n_k and a continuous observation r_k. An even more complicated case will involve an external input so that

$$x_k = \rho x_{k-1} + \alpha I_k + \varepsilon_k, \tag{1.9}$$

where αI_k is similar to the $B\mathbf{u}_k$ in (1.1), and x_k gives rise to a binary observation n_k and two continuous observations r_k and s_k. As we shall see, changes in the state equation primarily affect the predict step within the state estimation step. In contrast, changes in the observations mainly affect the update step.

Note that we mentioned the observation of *binary* and *continuous* features. When introducing the concept of physiological state estimation for the first time, we used the formulation

$$\mathbf{y}_k = C\mathbf{x}_k + \mathbf{v}_k \tag{1.10}$$

for the sensor measurements. In reality, this represents a very simple case, and the equations turn out to be similar to that of a Kalman filter. Sensor measurements in biomedical experiments can take many forms. They can take the form of binary-valued observations, continuous-valued observations, and spiking-type observations, to name a few. For instance, we may need to estimate the learning state of a macaque monkey in a behavioral experiment based on whether the monkey gets the answers correct or incorrect in different trials (a binary observation), how quickly the monkey responds in each trial (a continuous observation), and how electrical activity from a specific neuron varies over the trials (a spiking-type observation). These types of measurements result in filter equations that are more complicated than in the case of a Kalman filter. We will rely heavily on Bayes' rule to derive the mean and variance of x_k at the update step in each case.

While the state estimation step relies primarily on mean and variance calculations, the parameter estimation step relies mainly on derivatives. At the parameter estimation step, we take the derivatives of the probability terms (or equivalently, of the *log-likelihood* terms) to determine the model parameters. For instance, if we use the state equation in (1.8), we will need to derive ρ at the parameter estimation step. Moreover, we also need to determine the model parameters related to our observations. For instance, we may choose to model a continuous observation r_k as

$$r_k = \gamma_0 + \gamma_1 x_k + v_k, \tag{1.11}$$

where γ_0 and γ_1 are constant coefficients and $v_k \sim \mathcal{N}(0, \sigma_v^2)$ is sensor noise. The three parameters γ_0, γ_1, and σ_v^2 all need to be determined at the parameter estimation step. We could thus divide the parameter estimation step derivations into two parts. First, there will be the derivations for model parameters in the state equation (e.g.,

ρ, α, and σ_ε^2). And second, there will be the derivations corresponding to each of the observations (features). Choosing to include a continuous-valued observation in a state-space model will necessitate the determination of a certain set of model parameters. Adding a spiking-type observation necessitates a further set of model parameters. We will see examples of these in due course.

Having laid some of the basic groundwork, we will next proceed with our tutorial discussion of how to derive the state and parameter estimation step equations for several different physiological state-space models. Shown below is a list of the state-space models we will look at along with examples of where they have been applied:

- State-space model with one binary observation:
  - Behavioral learning [4]
  - Sympathetic arousal estimation using skin conductance signals [25, 26]
- State-space model with one binary and one continuous observation:
  - Behavioral learning [5]
  - Emotional valence estimation using electromyography (EMG) signals [27]
  - Seizure state estimation using scalp electroencephalography (EEG) signals [28]
- State-space model with one binary and two continuous observations:
  - Sympathetic arousal estimation using skin conductance signals [29]
  - Energy state estimation using blood cortisol concentrations [30]
- State-space model with one binary, two continuous, and a spiking-type observation:
  - Sympathetic arousal estimation using skin conductance and electrocardiography (EKG) signals [31]
- State-space model with one marked point process (MPP) observation:
  - Sympathetic arousal estimation using skin conductance signals [32]
- State-space model with one MPP and one continuous observation:
  - Energy state estimation using blood cortisol concentrations [33]
  - Sympathetic arousal estimation using skin conductance signals [33]

Wearable and smart healthcare technologies are likely to play a key role in the future [34, 35]. A number of the state-space models listed above have applicability to healthcare. For instance, patients suffering from emotional disorders, hormone dysregulation, or epileptic seizures could be fitted with wearable devices that implement some of the state-space models (and corresponding EM-based estimators) listed above for long-term care and monitoring. One of the advantages of the state-space framework is that it readily presents itself to the design of the *closed-loop control* necessary to correct deviation from healthy functioning. Consequently, state-space controllers can be designed to treat some of these disorders [36, 37]. Looking at the human body and brain from a control-theoretic perspective could also help design

bio-inspired controllers that are similar to its already built-in feedback control loops [38, 39]. The applications, however, are not just limited to healthcare monitoring, determining hidden psychological and cognitive states also has applications in fields such as neuromarketing [40], smart homes [41], and smart workplaces [42].

Excursus—A Brief Sketch of How the Kalman Filter Equations Can be Derived

Here we provide a brief sketch of how the Kalman filter equations can be derived. We will utilize an approach known as recursive least squares. The symbols used within this excursus are self-contained and should not be confused with the standard terminology that is used throughout the rest of this book.

Suppose we have a column vector of unknowns $\mathbf{x}$ and a column vector of measurements $\mathbf{y}_1$ that are related to each other through

$$\mathbf{y}_1 = A_1\mathbf{x} + \mathbf{e}_1, \tag{1.12}$$

where A_1 is a matrix and $\mathbf{e}_1 \sim \mathcal{N}(0, \Sigma_1)$ is noise (Σ_1 is the noise covariance matrix). In general, we may have more measurements than we have unknowns. Therefore, a solution to this system of equations is given by

$$\mathbf{x}_1 = (A_1^\intercal \Sigma_1^{-1} A_1)^{-1} A^\intercal \Sigma_1^{-1} \mathbf{y}_1, \tag{1.13}$$

where we have used $\mathbf{x}_1$ to denote that this solution is only based on the first set of measurements. Now suppose that we have another set of measurements $\mathbf{y}_2$ such that

$$\mathbf{y}_2 = A_2\mathbf{x} + \mathbf{e}_2, \tag{1.14}$$

where A_2 is a matrix and $\mathbf{e}_2 \sim \mathcal{N}(0, \Sigma_2)$. In theory, we could just concatenate all the values to form a single set of equations and solve for $\mathbf{x}$. However, this would result in a larger matrix inversion each time we get more data. Is there a better way? It turns out that we can use our previous solution $\mathbf{x}_1$ to obtain a better estimate $\mathbf{x}_2$ without having to solve everything again. If we assume that $\mathbf{e}_1$ and $\mathbf{e}_2$ are uncorrelated with each other, the least squares solution is given by

$$\mathbf{x}_2 = \left[\begin{pmatrix} A_1 \\ A_2 \end{pmatrix}^\intercal \begin{pmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{pmatrix}^{-1} \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}\right]^{-1} \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}^\intercal \begin{pmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix}^\intercal \tag{1.15}$$

$$= \left[\begin{pmatrix} A_1^\intercal & A_2^\intercal \end{pmatrix} \begin{pmatrix} \Sigma_1^{-1} & 0 \\ 0 & \Sigma_2^{-1} \end{pmatrix} \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}\right]^{-1} \begin{pmatrix} A_1^\intercal & A_2^\intercal \end{pmatrix} \begin{pmatrix} \Sigma_1^{-1} & 0 \\ 0 & \Sigma_2^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix}^\intercal \tag{1.16}$$

(continued)

$$= \left(A_1^\intercal \Sigma_1^{-1} A_1 + A_2^\intercal \Sigma_2^{-1} A_2\right)^{-1} \left(A_1^\intercal \Sigma_1^{-1} \mathbf{y}_1 + A_2^\intercal \Sigma_2^{-1} \mathbf{y}_2\right). \tag{1.17}$$

Let us see how this simplifies. We will begin by defining the term $P_1 = (A_1^\intercal \Sigma_1^{-1} A_1)^{-1}$. Now,

$$\mathbf{x}_1 = (A_1^\intercal \Sigma_1^{-1} A_1)^{-1} A^\intercal \Sigma_1^{-1} \mathbf{y}_1 \tag{1.18}$$

$$= P_1 A^\intercal \Sigma_1^{-1} \mathbf{y}_1 \tag{1.19}$$

$$\implies P_1^{-1} \mathbf{x}_1 = A^\intercal \Sigma_1^{-1} \mathbf{y}_1 \tag{1.20}$$

based on (1.13). Substituting P_1^{-1} for $A_1^\intercal \Sigma_1^{-1} A_1$ and $P_1^{-1}\mathbf{x}_1$ for $A^\intercal \Sigma_1^{-1} \mathbf{y}_1$ in (1.17), we obtain

$$\mathbf{x}_2 = \left(P_1^{-1} + A_2^\intercal \Sigma_2^{-1} A_2\right)^{-1} \left(P_1^{-1} \mathbf{x}_1 + A_2^\intercal \Sigma_2^{-1} \mathbf{y}_2\right). \tag{1.21}$$

We use the matrix inversion lemma to simplify this to

$$\mathbf{x}_2 = \left[P_1 - P_1 A_2^\intercal (\Sigma_2 + A_2 P_1 A_2^\intercal)^{-1} A_2 P_1\right] \left(P_1^{-1} \mathbf{x}_1 + A_2^\intercal \Sigma_2^{-1} \mathbf{y}_2\right). \tag{1.22}$$

We then perform the multiplication.

$$\begin{aligned} \mathbf{x}_2 = &\left[P_1 - P_1 A_2^\intercal (\Sigma_2 + A_2 P_1 A_2^\intercal)^{-1} A_2 P_1\right] P_1^{-1} \mathbf{x}_1 \\ &+ \left[P_1 - P_1 A_2^\intercal (\Sigma_2 + A_2 P_1 A_2^\intercal)^{-1} A_2 P_1\right] A_2^\intercal \Sigma_2^{-1} \mathbf{y}_2. \end{aligned} \tag{1.23}$$

For the time-being, we will ignore the terms on the right and make the substitution $K = P_1 A_2^\intercal (\Sigma_2 + A_2 P_1 A_2^\intercal)^{-1}$ for the term on the left. Therefore,

$$\begin{aligned} \mathbf{x}_2 = &\left(P_1 - K A_2 P_1\right) P_1^{-1} \mathbf{x}_1 \\ &+ \left[P_1 - P_1 A_2^\intercal (\Sigma_2 + A_2 P_1 A_2^\intercal)^{-1} A_2 P_1\right] A_2^\intercal \Sigma_2^{-1} \mathbf{y}_2 \end{aligned} \tag{1.24}$$

$$\begin{aligned} \mathbf{x}_2 = &P_1 P_1^{-1} \mathbf{x}_1 - K A_2 P_1 P_1^{-1} \mathbf{x}_1 \\ &+ \left[P_1 - P_1 A_2^\intercal (\Sigma_2 + A_2 P_1 A_2^\intercal)^{-1} A_2 P_1\right] A_2^\intercal \Sigma_2^{-1} \mathbf{y}_2 \end{aligned} \tag{1.25}$$

(continued)

$$\mathbf{x}_2 = \mathbf{x}_1 - KA_2\mathbf{x}_1 + \left[P_1 - P_1A_2^\intercal(\Sigma_2 + A_2P_1A_2^\intercal)^{-1}A_2P_1\right]A_2^\intercal\Sigma_2^{-1}\mathbf{y}_2. \tag{1.26}$$

When multiplying the terms on the right, we will define the term $Q = (\Sigma_2 + A_2P_1A_2^\intercal)^{-1}$. Making this substitution, we obtain

$$\mathbf{x}_2 = \mathbf{x}_1 - KA_2\mathbf{x}_1 + \left(P_1 - P_1A_2^\intercal QA_2P_1\right)A_2^\intercal\Sigma_2^{-1}\mathbf{y}_2 \tag{1.27}$$

$$\mathbf{x}_2 = \mathbf{x}_1 - KA_2\mathbf{x}_1 + P_1A_2^\intercal\Sigma_2^{-1}\mathbf{y}_2 - P_1A_2^\intercal QA_2P_1A_2^\intercal\Sigma_2^{-1}\mathbf{y}_2. \tag{1.28}$$

Here is where we will use a small trick. We will insert QQ^{-1} into the third term and then simplify.

$$\mathbf{x}_2 = \mathbf{x}_1 - KA_2\mathbf{x}_1 + P_1A_2^\intercal QQ^{-1}\Sigma_2^{-1}\mathbf{y}_2 - P_1A_2^\intercal QA_2P_1A_2^\intercal\Sigma_2^{-1}\mathbf{y}_2 \tag{1.29}$$

$$= \mathbf{x}_1 - KA_2\mathbf{x}_1 + P_1A_2^\intercal Q(Q^{-1} - A_2P_1A_2^\intercal)\Sigma_2^{-1}\mathbf{y}_2. \tag{1.30}$$

Since $Q = (\Sigma_2 + A_2P_1A_2^\intercal)^{-1}$, $Q^{-1} = \Sigma_2 + A_2P_1A_2^\intercal$. We will substitute this into (1.30) to obtain

$$\mathbf{x}_2 = \mathbf{x}_1 - KA_2\mathbf{x}_1 + P_1A_2^\intercal Q(\Sigma_2 + A_2P_1A_2^\intercal - A_2P_1A_2^\intercal)\Sigma_2^{-1}\mathbf{y}_2 \tag{1.31}$$

$$= \mathbf{x}_1 - KA_2\mathbf{x}_1 + P_1A_2^\intercal Q\Sigma_2\Sigma_2^{-1}\mathbf{y}_2 \tag{1.32}$$

$$= \mathbf{x}_1 - KA_2\mathbf{x}_1 + P_1A_2^\intercal Q\mathbf{y}_2. \tag{1.33}$$

Note that $P_1A_2^\intercal Q = P_1A_2^\intercal(\Sigma_2 + A_2P_1A_2^\intercal)^{-1} = K$. Therefore,

$$\mathbf{x}_2 = \mathbf{x}_1 - KA_2\mathbf{x}_1 + K\mathbf{y}_2 \tag{1.34}$$

$$= \mathbf{x}_1 + K(\mathbf{y}_2 - A_2\mathbf{x}_1). \tag{1.35}$$

What does the final equation mean? We simply take our previous solution $\mathbf{x}_1$, predict what $\mathbf{y}_2$ will be by multiplying it with A_2, calculate the prediction error $\mathbf{y}_2 - A_2\mathbf{x}_1$, and apply this correction to $\mathbf{x}_1$ based on the multiplication factor K. These equations, therefore, provide a convenient way to continually update $\mathbf{x}$ when we keep receiving more and more data.

Excursus—A Brief Sketch of How the EM Algorithm Works
Here we will provide a brief overview of how the EM algorithm works in the kind of state estimation problems that we shall see. Assume that we have a set of sensor measurements $\mathcal{Y} = \{y_1, y_2, \ldots, y_K\}$ and a set of unobserved states $\mathcal{X} = \{x_1, x_2, \ldots, x_K\}$ that we need to estimate. We also have the model parameters Θ that need to be determined.

Let us begin by asking the question as to how we can determine Θ. In general, we select Θ such that it maximizes the probability $p(\Theta|\mathcal{Y})$. Assuming that we do not have a particular preference for any of the Θ values, we can use Bayes' rule to instead select the Θ that maximizes $p(\mathcal{Y}|\Theta)$. Now,

$$p(\mathcal{Y}|\Theta) = \int_{\mathcal{X}} p(\mathcal{X} \cap \mathcal{Y}|\Theta) d\mathcal{X}. \tag{1.36}$$

We do not know what the true Θ is, but let us make a guess that it is $\hat{\Theta}$. Let us now introduce the term $p(\mathcal{X}|\mathcal{Y} \cap \hat{\Theta})$ into (1.36).

$$p(\mathcal{Y}|\Theta) = \int_{\mathcal{X}} \frac{p(\mathcal{X}|\mathcal{Y} \cap \hat{\Theta})}{p(\mathcal{X}|\mathcal{Y} \cap \hat{\Theta})} p(\mathcal{X} \cap \mathcal{Y}|\Theta) d\mathcal{X} \tag{1.37}$$

$$= \int_{\mathcal{X}} p(\mathcal{X}|\mathcal{Y} \cap \hat{\Theta}) \frac{p(\mathcal{X} \cap \mathcal{Y}|\Theta)}{p(\mathcal{X}|\mathcal{Y} \cap \hat{\Theta})} d\mathcal{X}. \tag{1.38}$$

Take a moment to look carefully at what the integral is doing. It is actually calculating the expected value of the fraction term with respect to $p(\mathcal{X}|\mathcal{Y} \cap \hat{\Theta})$. Taking the log on both sides, we have

$$\log\big[p(\mathcal{Y}|\Theta)\big] = \log\left[\int_{\mathcal{X}} p(\mathcal{X}|\mathcal{Y} \cap \hat{\Theta}) \frac{p(\mathcal{X} \cap \mathcal{Y}|\Theta)}{p(\mathcal{X}|\mathcal{Y} \cap \hat{\Theta})} d\mathcal{X}\right]. \tag{1.39}$$

Since $\log(\cdot)$ is a *concave* function, the following inequality holds true.

$$\log\big[p(\mathcal{Y}|\Theta)\big] \geq \int_{\mathcal{X}} p(\mathcal{X}|\mathcal{Y} \cap \hat{\Theta}) \log\left[\frac{p(\mathcal{X} \cap \mathcal{Y}|\Theta)}{p(\mathcal{X}|\mathcal{Y} \cap \hat{\Theta})}\right] d\mathcal{X} \tag{1.40}$$

$$\log\big[p(\mathcal{Y}|\Theta)\big] \geq \int_{\mathcal{X}} p(\mathcal{X}|\mathcal{Y} \cap \hat{\Theta}) \log\big[p(\mathcal{X} \cap \mathcal{Y}|\Theta)\big] d\mathcal{X} - \int_{\mathcal{X}} p(\mathcal{X}|\mathcal{Y} \cap \hat{\Theta}) \log\big[p(\mathcal{X}|\mathcal{Y} \cap \hat{\Theta})\big] d\mathcal{X}. \tag{1.41}$$

Recall that we set out to choose the Θ that maximized $p(\mathcal{Y}|\Theta)$, or that equivalently maximized $\log\big[p(\mathcal{Y}|\Theta)\big]$. Typically, we would approach this

(continued)

maximization by calculating the derivative of the probability term with respective to Θ, set it to $\mathbf{0}$, and then solve. For instance, if we had a continuous-valued observation r_k in our state-space model, we would have to take the derivatives with respect to γ_0, γ_1, and σ_v^2, set them each to 0, and solve. Look back at (1.41). Assume we were to calculate the derivative of the term on the right-hand side of the inequality with respective to Θ. Do you see that the second term *does not* contain Θ? In other words, the derivative would just treat the second term as a constant. If we had to determine γ_0, γ_1, and σ_v^2, for instance, they would only be present in the first term when taking derivatives. We can, therefore, safely ignore the second term. This leads to an important conclusion. If we need to determine the model parameters Θ by maximizing $\log\big[p(\mathcal{Y}|\Theta)\big]$, we only need to concentrate on maximizing

$$\int_{\mathcal{X}} p(\mathcal{X}|\mathcal{Y} \cap \hat{\Theta}) \log\big[p(\mathcal{X} \cap \mathcal{Y}|\Theta)\big] d\mathcal{X}. \tag{1.42}$$

We could equivalently write (1.42) as

$$\mathbb{E}_{\mathcal{X}|\mathcal{Y}\cap\hat{\Theta}}\Big[\log\big[p(\mathcal{X} \cap \mathcal{Y}|\Theta)\big]\Big] \tag{1.43}$$

since this is indeed an expected value. Do you now see the connection between what we have been discussing so far and the EM algorithm? In reality, what we are doing at the state estimation step is calculating $\mathbb{E}[\mathcal{X}|\mathcal{Y} \cap \hat{\Theta}]$. At the parameter estimation step, we calculate the partial derivatives of the expected value of $\log\big[p(\mathcal{X} \cap \mathcal{Y}|\Theta)\big]$ with respect to all of the model parameters. During the actual implementation of the EM algorithm, we keep alternating between the two steps until the model parameters converge. At this point, we have reached one of the localized maximum values of $\mathbb{E}_{\mathcal{X}|\mathcal{Y}\cap\hat{\Theta}}\Big[\log\big[p(\mathcal{X} \cap \mathcal{Y}|\Theta)\big]\Big]$.

Chapter 2
Some Useful Statistical Results

The EM algorithms for state estimation that we consider in this book rely on basic concepts in statistics. In this chapter, we will review some results that will come in useful later on. Many of the concepts are introductory and only require a basic knowledge of probability and statistics. If you are already familiar with what is discussed here, feel free to skip ahead.

2.1 Basic Concepts Related to Mean and Variance

Shown below are some basic statistical results related to mean and variance that will be helpful when deriving the EM algorithm equations.

Basic Statistical Results—Part A
Given the random variables X_k and Z_k, and the constant values ρ and α, the following results hold true for mean and variance. We use $\mathbb{E}[\cdot]$, $V(\cdot)$, and $Cov(\cdot)$ to denote the mean or expected value, the variance, and covariance, respectively.

$$\mathbb{E}[X_k + Z_k] = \mathbb{E}[X_k] + \mathbb{E}[Z_k] \tag{2.1}$$

$$\mathbb{E}[X_k + \alpha] = \mathbb{E}[X_k] + \alpha \tag{2.2}$$

$$\mathbb{E}[\rho X_k] = \rho \mathbb{E}[X_k] \tag{2.3}$$

$$V(X_k + Z_k) = V(X_k) + V(Z_k) + 2Cov(X_k, Z_k) \tag{2.4}$$

(continued)

D. S. Wickramasuriya, R. T. Faghih, *Bayesian Filter Design for Computational Medicine*, https://doi.org/10.1007/978-3-031-47104-9_2

$$V(X_k + \alpha) = V(X_k) \tag{2.5}$$

$$V(\rho X_k) = \rho^2 V(X_k). \tag{2.6}$$

2.2 Basic Statistical Results Required for Deriving the Update Equations in the State Estimation Step

We will also require two results based on Bayes' rule and Gaussian distributions, respectively, when deriving the update equations in the state estimation step. The two results are shown below:

- Result 1:

$$P(A|B \cap C) = \frac{P(B|A \cap C)P(A|C)}{P(B|C)}.$$

  We will consider the derivation of this result in two steps:

  – Step 1:

$$P(A|B \cap C) = \frac{P(A \cap B \cap C)}{P(B \cap C)} \tag{2.7}$$

$$= \frac{P(A \cap B \cap C)}{P(B \cap C)} \times \frac{P(C)}{P(C)} \tag{2.8}$$

$$= \frac{P(A \cap B \cap C)}{P(C)} \times \frac{1}{\frac{P(B \cap C)}{P(C)}} \tag{2.9}$$

$$= \frac{P(A \cap B|C)}{P(B|C)}. \tag{2.10}$$

  – Step 2:

$$P(A \cap B|C) = \frac{P(A \cap B \cap C)}{P(C)} \tag{2.11}$$

$$= \frac{P(A \cap B \cap C)}{P(C)} \times \frac{P(A \cap C)}{P(A \cap C)} \tag{2.12}$$

$$= \frac{P(A \cap B \cap C)}{P(A \cap C)} \times \frac{P(A \cap C)}{P(C)} \tag{2.13}$$

$$= P(B|A \cap C)P(A|C). \tag{2.14}$$

We will substitute the result for $P(A \cap B|C)$ in (2.14) and put it into (2.10) to obtain

$$P(A|B \cap C) = \frac{P(A \cap B|C)}{P(B|C)} = \frac{P(B|A \cap C)P(A|C)}{P(B|C)}. \tag{2.15}$$

Basic Statistical Results—Part B
Letting $A = X_k$, $B = Y_k$, and $C = Y_{1:k-1}$, we can use the result just shown above to obtain

$$P(X_k|Y_{1:k}) = P(X_k|Y_k, Y_{1:k-1}) = \frac{P(Y_k|X_k, Y_{1:k-1})P(X_k|Y_{1:k-1})}{P(Y_k|Y_{1:k-1})}. \tag{2.16}$$

Recall that we split our state estimation into two steps: the predict step and the update step. At the predict step, we derive an estimate for x_k given that we have not yet observed the sensor reading y_k. This estimate is actually based on $P(X_k|Y_{1:k-1})$ since information only available until time index $(k-1)$ is used to derive it. At the update step, we improve the predict step estimate based on $P(X_k|Y_{1:k-1})$ to now include information from the new sensor measurement y_k, i.e., we make use of y_k to obtain a new estimate based on $P(X_k|Y_{1:k})$. The result in (2.16) will come in very useful at the update step.

- Result 2:
  The mean and variance of a Gaussian random variable can be obtained by taking the derivatives of the exponent term of its probability density function (PDF).
  Consider $X \sim \mathcal{N}(\mu, \sigma^2)$. The PDF of X is given by

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{qx} \text{ where } q = \frac{-(x-\mu)^2}{2\sigma^2}. \tag{2.17}$$

  To obtain the mean of X, we take the derivative of q and set it to 0 to determine where the maximum value occurs.

$$\frac{dq}{dx} = \frac{-2(x-\mu)}{2\sigma^2} = 0 \tag{2.18}$$

$$\implies x = \mu. \tag{2.19}$$

  Therefore, the mean value occurs at the location for x at which the derivative of the exponent term is equal to 0.

We next consider the variance. The second derivative of q with respect to x is

$$\frac{d^2q}{dx^2} = \frac{-1}{\sigma^2}. \tag{2.20}$$

And therefore, the variance is given by

$$\implies \sigma^2 = -\left(\frac{d^2q}{dx^2}\right)^{-1}. \tag{2.21}$$

Basic Statistical Results—Part C
In all the derivations of the state estimation step update equations, we will assume that the density functions are approximately Gaussian. We will also make use of what we have just shown: (i) the mean value is given by the location at which the first derivative of the exponent term is equal to 0; (ii) the variance is given by the negative inverse of the second derivative of the exponent term.

2.3 General Observations Related to Gaussian Random Variables

In general, for a set of independent Gaussian random variables $Z_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$, the following holds true.

$$\sum_i a_i Z_i \sim \mathcal{N}\left(\sum_i a_i \mu_i, \sum_i a_i^2 \sigma_i^2\right), \tag{2.22}$$

where the a_i's are constant terms. Also, adding a constant term to a Gaussian random variable will cause it to remain Gaussian but have a shifted mean and unchanged variance. This can be verified from first principles (change of variables formula).

Basic Statistical Results—Part D
In general, for a set of independent Gaussian random variables $Z_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$,

$$\sum_i a_i Z_i \sim \mathcal{N}\left(\sum_i a_i \mu_i, \sum_i a_i^2 \sigma_i^2\right). \tag{2.23}$$

Chapter 3
State-Space Model with One Binary Observation

In this chapter, we will consider a state-space model where a single state variable x_k gives rise to binary observations. We will see how the state and parameter estimation equations are derived for this case. However, prior to deriving any of the equations, we will first look at two example scenarios where the need for such a model arises.

We human beings learn. We start learning since the time we were born, and learning continues thereafter as a life-long process. How exactly do we learn? And how do animals learn? These are interesting problems that scientists have investigated for years. One of the problems that arises in learning experiments with animal models is determining when an animal is considered to have learned something. For instance, suppose that a macaque monkey needs to learn how to correctly identify a particular visual target shown on a computer screen. The monkey may receive a reward for every correct answer. Similarly, a rat may have to learn to how to recognize an audio cue to receive a reward in a maze (Fig. 3.1). How could we know that the animal has actually learned? This is an interesting question. We could, for instance, come up with heuristic rules such as stating that the animal has indeed learned when five consecutive correct answers (or some other number) are recorded. But could something more systematic be developed? This problem is what motivated the work in [4]. Here, learning was characterized using a state-space model. Since correct and incorrect are the only possible trial outcomes, the observations are binary-valued. Moreover, rather than just deciding whether the animal has learned or not yet learned, the objective was to estimate a *continuous* learning state x_k based on the sequence of binary responses n_k. When learning has not yet occurred, more incorrect responses occur in the trials and x_k remains

Supplementary Information The online version contains supplementary material available at (https://doi.org/10.1007/978-3-031-47104-9_3).

GitHub https://github.com/computational-medicine-lab/A-Tutorial-on-the-Design-of-Bayesian-Filters/tree/main/one_bin

D. S. Wickramasuriya, R. T. Faghih, *Bayesian Filter Design for Computational Medicine*, https://doi.org/10.1007/978-3-031-47104-9_3

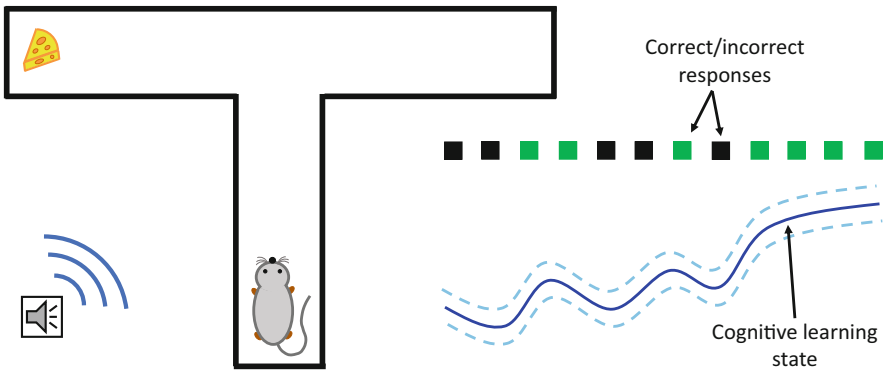

Fig. 3.1 A rat in a T-maze experiment with binary-valued correct/incorrect responses. Binary-valued correct/incorrect responses can be used to estimate the cognitive learning state of a rat based on its responses in successive trials. The model was used in [4] for this purpose where the rat had to learn to recognize which direction to proceed in based on an audio cue

low. However, as the animal begins to learn, more correct responses occur and x_k increases. Thus it is possible to see how learning continuously progresses over successive trials.

The second example relates to emotions and the nervous system. We primarily sweat to maintain internal body temperature. However, tiny bursts of sweat are also released in response to psychologically arousing stimuli. These variations in sweat secretions cause changes in the conductivity of the skin and can be picked up easily by skin conductance sensors. Since the sweat glands are innervated by nerve fibers belonging to the sympathetic branch of the autonomic nervous system [43], a skin conductance signal becomes a sensitive index of *sympathetic arousal* [44]. Now a skin conductance signal comprises a slow-varying *tonic* component on top of which a faster-varying *phasic* component is superimposed [45, 46]. The phasic component consists of what are known as skin conductance responses (SCRs). These SCRs have characteristic bi-exponential shapes. Each of these SCRs can be thought of as being produced by a single burst of neuroelectric activity to the sweat glands [47]. It is these phasic SCRs that give a skin conductance signal its "spikey" appearance (Fig. 3.2). A deconvolution algorithm can be used to recover the bursts of neural activity underlying a skin conductance signal [47–50]. Importantly, the occurrence of these neural impulses is related to a person's arousal level. In particular, the higher the underlying sympathetic arousal, the higher the rate at which neural impulses to the sweat glands (or SCRs) occur [51]. Thus the same state-space model with binary observations based on neural impulses to the sweat glands was used in [26] to estimate sympathetic arousal. By tracking the occurrence of the impulses n_k, a person's arousal state could be estimated over time.

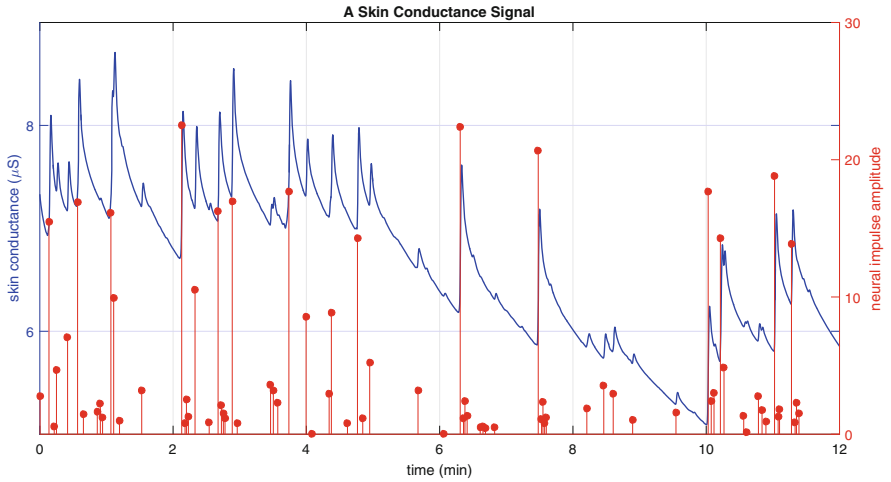

Fig. 3.2 A deconvolved skin conductance signal. A skin conductance signal comprises both a tonic and phasic component. The neural impulses underlying phasic variations can be extracted via deconvolution. The figure depicts a skin conductance signal (blue) and the sequence of neural impulses that underlie its phasic variations (red). From [32], used under Creative Commons CC-BY license

3.1 Deriving the Predict Equations in the State Estimation Step

Let us now consider the state-space model itself. For simplicity, we will also not use upper case letters for the unknowns although they are indeed random variables. Instead, we will follow the more familiar notation for state-space control systems with lower case letters. Let us begin by assuming that x_k evolves with time following a random walk.

$$x_k = x_{k-1} + \varepsilon_k, \tag{3.1}$$

where the process noise term $\varepsilon_k \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ is independent of any of the x_k values.

For now, let us not think of (3.1) as being the state equation in a control system. Instead, let us just consider (3.1) purely as a relationship between three random variables. Supposing we only had this equation and had to determine x_k, what would be the best guess that we could come up with and how uncertain would we be about it? Our best estimate for x_k would be its mean, and the uncertainty associated with it would be its variance. We will use the basic formulas in (2.1)–(2.6) to determine the mean and variance of x_k. We will first derive the mean.

$$\mathbb{E}[x_k] = \mathbb{E}[x_{k-1} + \varepsilon_k] \tag{3.2}$$

$$= \mathbb{E}[x_{k-1}] + \mathbb{E}[\varepsilon_k] \text{ using (2.1)} \tag{3.3}$$

$$= \mathbb{E}[x_{k-1}] \text{ since } \mathbb{E}[\varepsilon_k] = 0 \tag{3.4}$$

$$\therefore \mathbb{E}[x_k] = x_{k-1|k-1}, \tag{3.5}$$

where we have used the notation $x_{k-1|k-1}$ to denote the expected value $\mathbb{E}[x_{k-1}]$. In a typical state-space control system, $x_{k-1|k-1}$ represents the best estimate for x_{k-1} given that we have observed all the sensor measurements up to time index $(k-1)$. We will also use the notation $\mathbb{E}[x_k] = x_{k|k-1}$ to denote the mean state estimate at time index k, given that we have only observed the sensor readings until time index $(k-1)$.

Next we will derive the uncertainty or variance of x_k using the same basic formulas.

$$V(x_k) = V(x_{k-1} + \varepsilon_k) \tag{3.6}$$

$$= V(x_{k-1}) + V(\varepsilon_k) + 2Cov(x_{k-1}, \varepsilon_k) \text{ using (2.4)} \tag{3.7}$$

$$= V(x_{k-1}) + V(\varepsilon_k) \text{ since } \varepsilon_k \text{ is uncorrelated with any of the } x_k \text{ terms} \tag{3.8}$$

$$\therefore V(x_k) = \sigma^2_{k-1|k-1} + \sigma^2_{\varepsilon}, \tag{3.9}$$

where we have used the notation $\sigma^2_{k-1|k-1}$ to denote the variance $V(x_{k-1})$. Again, in a typical state-space control system, $\sigma^2_{k-1|k-1}$ represents the uncertainty or variance of x_{k-1} given that we have observed all the sensor readings up to time index $(k-1)$. Just like in the case of the mean, we will use the notation $V(x_k) = \sigma^2_{k|k-1}$ to denote that this is the variance estimate at time index k, given that we have only observed the sensor readings until time index $(k-1)$. Therefore, our predict equations in the state estimation step are

$$x_{k|k-1} = x_{k-1|k-1} \tag{3.10}$$

$$\sigma^2_{k|k-1} = \sigma^2_{k-1|k-1} + \sigma^2_{\varepsilon}. \tag{3.11}$$

From our knowledge of Gaussian distributions in (2.23), we also know that x_k is Gaussian distributed since x_{k-1} and ε_k are Gaussian distributed and independent of each other. Since we have just derived the mean and variance of x_k, we can state that

$$p(x_k|n_{1:k-1}) = \frac{1}{\sqrt{2\pi\sigma^2_{k|k-1}}} e^{\frac{-(x_k - x_{k|k-1})^2}{2\sigma^2_{k|k-1}}}, \tag{3.12}$$

where the conditioning on $n_{1:k-1}$ indicates that we have observed the sensor measurements up to time index $(k-1)$. What happens when we observe measurement n_k at time index k? We will see how our estimates $x_{k|k-1}$ and $\sigma^2_{k|k-1}$ can be improved/updated once we observe n_k in the next section.

When x_k evolves with time following $x_k = x_{k-1} + \varepsilon_k$, the predict equations in the state estimation step are

$$x_{k|k-1} = x_{k-1|k-1} \tag{3.13}$$

$$\sigma^2_{k|k-1} = \sigma^2_{k-1|k-1} + \sigma^2_{\varepsilon}. \tag{3.14}$$

3.2 Deriving the Update Equations in the State Estimation Step

The binary observations n_k that we consider here could be in the form of correct/incorrect responses in a behavioral experiment, neural impulses in a skin conductance signal, hormone pulses, etc. Let us assume that x_k is related to the probability p_k with which the binary events occur through

$$p_k = \frac{1}{1 + e^{-(\beta_0 + x_k)}}, \tag{3.15}$$

where β_0 is a constant. Here, $p_k = P(n_k = 1)$ and $(1 - p_k) = P(n_k = 0)$. Equation (3.15) depicts what is known as a *sigmoid* relationship. Accordingly, the higher x_k is, the higher will be p_k. In other words, the higher x_k is, the higher the probability of 1's occurring in the observations.

At this point, we need to note an important result concerning the derivative of the sigmoid function.

$$\frac{dp_k}{dx_k} = \frac{(-1)}{[1 + e^{-(\beta_0 + x_k)}]^2} \times e^{-(\beta_0 + x_k)} \times (-1) \tag{3.16}$$

$$= \frac{1}{1 + e^{-(\beta_0 + x_k)}} \times \left[\frac{e^{-(\beta_0 + x_k)}}{1 + e^{-(\beta_0 + x_k)}} \right] \tag{3.17}$$

$$= \frac{1}{1 + e^{-(\beta_0 + x_k)}} \times \left[\frac{1 + e^{-(\beta_0 + x_k)} - 1}{1 + e^{-(\beta_0 + x_k)}} \right] \tag{3.18}$$

$$= \frac{1}{1 + e^{-(\beta_0 + x_k)}} \times \left[1 - \frac{1}{1 + e^{-(\beta_0 + x_k)}} \right] \tag{3.19}$$

$$= p_k(1 - p_k). \tag{3.20}$$

Now the occurrence of $n_k = 0$ or $n_k = 1$ follows a *Bernoulli distribution*. Therefore,

$$p(n_k|x_k) = p_k^{n_k}(1-p_k)^{1-n_k} \tag{3.21}$$

$$= \left[\frac{1}{1+e^{-(\beta_0+x_k)}}\right]^{n_k}\left[1-\frac{1}{1+e^{-(\beta_0+x_k)}}\right]^{1-n_k}. \tag{3.22}$$

We will also utilize another useful result here. For a positive number a, $a = e^{\log(a)}$ (this can be easily verified by taking the log value on both sides). We can use this to express $p(n_k|x_k)$ as shown below.

$$p(n_k|x_k) = p_k^{n_k}(1-p_k)^{1-n_k} \tag{3.23}$$

$$= e^{\log\left[p_k^{n_k}(1-p_k)^{1-n_k}\right]} \tag{3.24}$$

$$= e^{\log\left[p_k^{n_k}\right]+\log\left[(1-p_k)^{1-n_k}\right]} \tag{3.25}$$

$$= e^{n_k\log(p_k)+(1-n_k)\log(1-p_k)} \tag{3.26}$$

$$= e^{n_k\log\left(\frac{p_k}{1-p_k}\right)+\log(1-p_k)}. \tag{3.27}$$

Now assume that we just observed n_k. What would be our best estimate of x_k given that we have observed $n_{1:k}$? In other words, what is $p(x_k|n_{1:k})$, and how can we derive its mean and variance? We can use the result in (2.16) to determine what $p(x_k|n_{1:k})$ is.

$$p(x_k|n_{1:k}) = p(x_k|n_k, n_{1:k-1}) = \frac{p(n_k|x_k, n_{1:k-1})p(x_k|n_{1:k-1})}{p(n_k|n_{1:k-1})}. \tag{3.28}$$

Let us consider the terms in the numerator. Now $p(n_k|x_k, n_{1:k-1}) = p(n_k|x_k)$ since we have an explicit relationship between n_k and x_k as shown in (3.15), which makes the additional conditioning on the history $n_{1:k-1}$ irrelevant. We know what $p(n_k|x_k)$ is based on (3.26). We also know what $p(x_k|n_{1:k-1})$ is based on (3.12).

We now need to determine the mean and variance of $p(x_k|n_{1:k})$. To do so, we will assume that it is approximately Gaussian distributed. Recall from the earlier results in (2.19) and (2.21) that the mean and variance of a Gaussian distribution can be derived from its exponent term alone. Therefore, we only need to consider the exponent of $p(x_k|n_{1:k})$ and can ignore the other terms. We will therefore substitute the terms for $p(n_k|x_k)$ and $p(x_k|n_{1:k-1})$ in (3.26) and (3.12), respectively, into (3.28).

$$p(x_k|n_{1:k}) \propto p(n_k|x_k)p(x_k|n_{1:k-1}) \propto e^{n_k\log(p_k)+(1-n_k)\log(1-p_k)} \times e^{\frac{-(x_k-x_{k|k-1})^2}{2\sigma^2_{k|k-1}}}. \tag{3.29}$$

Taking the log on both sides and labeling it as q, we have

$$q = \log[p(x_k|n_{1:k})] = n_k \log(p_k) + (1 - n_k) \log(1 - p_k) - \frac{(x_k - x_{k|k-1})^2}{2\sigma^2_{k|k-1}} + \text{ constant.} \tag{3.30}$$

This equation provides us the exponent of $p(x_k|n_{1:k})$, which we will use to derive the mean and variance. We can obtain the mean by taking the first derivative of the exponent and then solving for the location where it is 0. Likewise the variance is given by the negative inverse of the second derivative.

Let us first proceed with calculating the mean. We will make use of the formula for the derivative of p_k in (3.20).

$$\frac{dq}{dx_k} = n_k \frac{1}{p_k} \frac{dp_k}{dx_k} + (1 - n_k) \frac{1}{(1 - p_k)} \frac{d}{dx}(1 - p_k) - \frac{2(x_k - x_{k|k-1})}{2\sigma^2_{k|k-1}} = 0 \tag{3.31}$$

$$n_k \frac{1}{p_k} p_k(1 - p_k) - (1 - n_k) \frac{1}{(1 - p_k)} p_k(1 - p_k) - \frac{(x_k - x_{k|k-1})}{\sigma^2_{k|k-1}} = 0 \tag{3.32}$$

$$n_k(1 - p_k) - (1 - n_k) p_k - \frac{(x_k - x_{k|k-1})}{\sigma^2_{k|k-1}} = 0 \tag{3.33}$$

$$n_k - p_k - \frac{(x_k - x_{k|k-1})}{\sigma^2_{k|k-1}} = 0 \tag{3.34}$$

$$n_k - p_k = \frac{(x_k - x_{k|k-1})}{\sigma^2_{k|k-1}} \tag{3.35}$$

$$x_k = x_{k|k-1} + \sigma^2_{k|k-1}(n_k - p_k). \tag{3.36}$$

This equation gives us the mean of x_k, which is now our *new* best estimate given that we have observed all the data up to time index k. We will call this new mean $x_{k|k}$. It is an improvement over $x_{k|k-1}$, which did not include information from the latest observation. Since

$$p_k = \frac{1}{1 + e^{-(\beta_0 + x_k)}}, \tag{3.37}$$

the x_k term appears on both sides of (3.36). Therefore, the equation has to be solved numerically (e.g., using Newton's method). To make this dependency explicit, we will use the notation $p_{k|k}$ and express the mean as

$$x_{k|k} = x_{k|k-1} + \sigma^2_{k|k-1}(n_k - p_{k|k}). \tag{3.38}$$

We will next derive the variance. Now the first derivative of the exponent simplified to

$$\frac{dq}{dx_k} = n_k - p_k - \frac{(x_k - x_{k|k-1})}{\sigma^2_{k|k-1}}. \tag{3.39}$$

The second derivative yields

$$\frac{d^2q}{dx_k^2} = -p_k(1 - p_k) - \frac{1}{\sigma^2_{k|k-1}}. \tag{3.40}$$

Based on our knowledge of how variance can be derived from the exponent term in a Gaussian distribution, the uncertainty or variance associated with our new state estimate is

$$\sigma^2_{k|k} = -\left(\frac{d^2q}{dx_k^2}\right)^{-1} = \left[\frac{1}{\sigma^2_{k|k-1}} + p_k(1 - p_k)\right]^{-1}. \tag{3.41}$$

Again, we will make the dependence of p_k on $x_{k|k}$ explicit and state

$$\sigma^2_{k|k} = \left[\frac{1}{\sigma^2_{k|k-1}} + p_{k|k}(1 - p_{k|k})\right]^{-1}. \tag{3.42}$$

When x_k gives rise to a single binary observation n_k, the update equations in the state estimation step are

$$x_{k|k} = x_{k|k-1} + \sigma^2_{k|k-1}(n_k - p_{k|k}) \tag{3.43}$$

$$\sigma^2_{k|k} = \left[\frac{1}{\sigma^2_{k|k-1}} + p_{k|k}(1 - p_{k|k})\right]^{-1}. \tag{3.44}$$

3.3 Smoothing in the State Estimation Step

Although we previously stated that the state estimation step primarily consisted of the predict and update steps, in reality, there is a third step that we follow. The

equations for this third step, however, do not vary much depending on the state-space model and consequently do not require re-derivations every time we have a new model. In fact, as we shall see, there is only one case where we need to make changes to this third step. Now we first perform the predict, update, predict, update, ... steps in turn for $k = 1, 2, \ldots, K$ to determine x_k at each point in time. After coming to the end, we *reverse* direction and obtain a set of smoothened mean and variance estimates. The equations for this *backward smoother* are

$$A_k \triangleq \frac{\sigma^2_{k|k}}{\sigma^2_{k+1|k}} \tag{3.45}$$

$$x_{k|K} = x_{k|k} + A_k(x_{k+1|K} - x_{k+1|k}) \tag{3.46}$$

$$\sigma^2_{k|K} = \sigma^2_{k|k} + A^2_k(\sigma^2_{k+1|K} - \sigma^2_{k+1|k}). \tag{3.47}$$

The only change that occurs in these equations is if there is a forgetting factor ρ in the state equation (e.g., $x_k = \rho x_{k-1} + \varepsilon_k$). In this case, we would have

$$A_k \triangleq \rho \frac{\sigma^2_{k|k}}{\sigma^2_{k+1|k}}. \tag{3.48}$$

Since we reverse direction making use of *all* the data through $k = 1, 2, \ldots, K$ to obtain the smoothened mean and variance estimates, we use the notation $x_{k|K}$ and $\sigma^2_{k|K}$ to denote their values. These new estimates turn out to be *smoother* since we now determine x_k not just based on $k = 1, 2, \ldots, k$ (what we have observed up to that point), but rather on $k = 1, 2, \ldots, K$ (all what we have observed).

We will also make a further observation. We need to note that $x_{k|K}$ and $\sigma^2_{k|K}$ can be formally expressed as

$$x_{k|K} = \mathbb{E}[x_k | n_{1:K}, \Theta] \tag{3.49}$$

$$\sigma^2_{k|K} = V(x_k | n_{1:K}, \Theta), \tag{3.50}$$

where Θ represents all the model parameters. In the case of our current state-space model, the only unknown model parameter is σ^2_ε (and β_0, but we will assume that this is calculated differently). Why is the expected value conditioned on Θ? Recall that the EM algorithm consists of the state and parameter estimation steps. At the state estimation step, we assume that we *know* all the model parameters and proceed with calculating x_k. Mathematically, we could express this knowledge of the model parameters in terms of conditioning on Θ. In reality, we could also have expressed $x_{k|k-1}$ and $x_{k|k}$ (and the variances) in a similar manner, i.e.,

$$x_{k|k-1} = \mathbb{E}[x_k | n_{1:k-1}, \Theta] \tag{3.51}$$

$$x_{k|k} = \mathbb{E}[x_k | n_{1:k}, \Theta]. \tag{3.52}$$

Finally, we also need to note that we often require not only $\mathbb{E}[x_k|n_{1:K}, \Theta]$, but also $\mathbb{E}[x_k^2|n_{1:K}, \Theta]$ and $\mathbb{E}[x_k x_{k+1}|n_{1:K}, \Theta]$ when we move on to the parameter estimation step. Making use of the state-space covariance algorithm [52], these values turn out to be

$$\mathbb{E}[x_k^2|n_{1:K}, \Theta] = U_k = x_{k|K}^2 + \sigma_{k|K}^2 \tag{3.53}$$

$$\mathbb{E}[x_k x_{k+1}|n_{1:K}, \Theta] = U_{k,k+1} = x_{k|K} x_{k+1|K} + A_k \sigma_{k+1|K}^2, \tag{3.54}$$

where we have defined the two new terms U_k and $U_{k,k+1}$.

3.4 Deriving the Parameter Estimation Step Equations

Recall our earliest discussion of the EM algorithm. To describe how it functioned, we assumed the simple state-space model

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k \tag{3.55}$$

$$\mathbf{y}_k = C\mathbf{x}_k. \tag{3.56}$$

We stated that, at our state estimation step, we would assume that we knew A, B, and C and then determine the best estimates for $\mathbf{x}_k$. The state estimation step consists of the predict step, the update step, and the smoothing step that we perform at then end. At the predict step, we make a prediction for $\mathbf{x}_k$ using the state equation based on the past history of values. At the update step, we improve this prediction by making use of the sensor reading $\mathbf{y}_k$ that we just observed. After proceeding through the predict, update, predict, update... steps, we finally reverse direction and perform smoothing. We primarily make use of the ideas of mean and variance at the state estimation step. It is after performing the state estimation step that we proceed to the parameter estimation step where we make use of the $\mathbf{x}_k$ estimates and determine A, B, and C. We select A, B, and C to maximize a particular probability. This probability is the joint density of all our $\mathbf{x}_k$ and $\mathbf{y}_k$ values. We also stated that, in reality, it was not strictly the probability that we maximize, but rather the *expected* value or mean of its log. Do you now see why the state estimation step involved calculating the expected values of x_k?

Let us now consider the joint probability term whose expected value of the log we need to maximize. It is

$$p(x_{1:K} \cap y_{1:K}|\Theta) = p(y_{1:K}|x_{1:K}, \Theta)p(x_{1:K}|\Theta). \tag{3.57}$$

Since we only observe a single binary variable, we have $y_k = n_k$. Therefore,

$$p(x_{1:K} \cap n_{1:K}|\Theta) = p(n_{1:K}|x_{1:K}, \Theta)p(x_{1:K}|\Theta). \tag{3.58}$$

We will first consider $p(x_{1:K}|\Theta)$. What would be the total probability of all the x_k values if we only knew the model parameters Θ? In other words, if we had no sensor readings n_k, what would be the probability of our x_k values? To calculate this, we would *only* be able to make use of the state equation, but not the output equation. This probability is

$$\begin{aligned} p(x_{1:K}|\Theta) &= p(x_1|\Theta) \times p(x_2|x_1, \Theta) \\ &\quad \times p(x_3|x_1, x_2, \Theta) \times \ldots \times p(x_K|x_1, x_2, \ldots, x_{K-1}, \Theta) \qquad (3.59) \\ &= \prod_{k=1}^{K} \frac{1}{\sqrt{2\pi\sigma_\varepsilon^2}} e^{\frac{-(x_k - x_{k-1})^2}{2\sigma_\varepsilon^2}}. \qquad (3.60) \end{aligned}$$

Note that in the case of each term x_k, x_{k-1} contains within it the history needed to get to it. Let us take the log of this value and label it $\tilde{Q}$.

$$\tilde{Q} = \frac{-K}{2} \log\left(2\pi\sigma_\varepsilon^2\right) - \sum_{k=1}^{K} \frac{(x_k - x_{k-1})^2}{2\sigma_\varepsilon^2}. \qquad (3.61)$$

Now the only model parameter we need to determine is σ_ε^2 (ignoring β_0). It turns out that σ_ε^2 only shows up in this term involving $p(x_{1:K}|\Theta)$ and not in the term involving $p(n_{1:K}|x_{1:K}, \Theta)$. Let us now take the expected value of $\tilde{Q}$ and label it Q.

$$Q = \frac{-K}{2} \log\left(2\pi\sigma_\varepsilon^2\right) - \sum_{k=1}^{K} \frac{\mathbb{E}\left[(x_k - x_{k-1})^2\right]}{2\sigma_\varepsilon^2}. \qquad (3.62)$$

What do we need to do at the parameter estimation step to determine σ_ε^2? We simply need to take the derivative of Q with respect to σ_ε^2, set it to 0, and solve. But the expected value we need should be calculated conditioned on knowing Θ and having observed $n_{1:K}$ (i.e., we need $\mathbb{E}[x_k|n_{1:K}, \Theta]$). Do you now see why we expressed $x_{k|K}$ and $\sigma_{k|K}^2$ in the way that we did in (3.49) and (3.50)?

3.4.1 Deriving the Process Noise Variance

While it is possible to determine the starting state x_0 as a separate parameter, we follow one of the options in [4, 5] and set $x_0 = x_1$. This permits some bias at the beginning. Therefore,

$$Q = \frac{-K}{2} \log\left(2\pi\sigma_\varepsilon^2\right) - \sum_{k=2}^{K} \frac{\mathbb{E}\left[(x_k - x_{k-1})^2\right]}{2\sigma_\varepsilon^2}. \qquad (3.63)$$

We will follow this method of setting $x_0 = x_1$ in all of our parameter estimation step derivations. We take the partial derivative of Q with respect to σ_ε^2 and set it to 0 to solve for the parameter estimation step update.

$$\frac{\partial Q}{\partial \sigma_\varepsilon^2} = \frac{-K}{2\sigma_\varepsilon^2} + \frac{1}{2\sigma_\varepsilon^4} \sum_{k=2}^{K} \mathbb{E}\Big[(x_k - x_{k-1})^2\Big] = 0 \tag{3.64}$$

$$\implies \sigma_\varepsilon^2 = \frac{1}{K} \sum_{k=2}^{K} \Big\{ \mathbb{E}\big[x_k^2\big] - 2\mathbb{E}\big[x_k x_{k-1}\big] + \mathbb{E}\big[x_{k-1}^2\big] \Big\} \tag{3.65}$$

$$= \frac{1}{K} \Bigg\{ \sum_{k=2}^{K} U_k - 2 \sum_{k=1}^{K-1} U_{k,k+1} + \sum_{k=1}^{K-1} U_k \Bigg\}. \tag{3.66}$$

The parameter estimation step update for σ_ε^2 when x_k evolves with time following $x_k = x_{k-1} + \varepsilon_k$ is

$$\sigma_\varepsilon^2 = \frac{1}{K} \Bigg\{ \sum_{k=2}^{K} U_k - 2 \sum_{k=1}^{K-1} U_{k,k+1} + \sum_{k=1}^{K-1} U_k \Bigg\}. \tag{3.67}$$

3.5 MATLAB Examples

In this book, we also provide a set of MATLAB code examples that implement the EM algorithms described in each chapter. The code examples are organized into the folder structure shown below:

- one_bin\
  - sim\
    - data_one_bin.mat
    - filter_one_bin.m
  - expm\
    - expm_data_one_bin.mat
    - expm_filter_one_bin.m
- one_mpp\
  - sim\
    - data_one_mpp.mat

filter_one_mpp.m

expm\

expm_data_one_mpp.mat
expm_filter_one_mpp.m

- one_bin_two_cont\

  ...

- one_mpp_one_cont\

  ...

- ...

In the case of each state-space model, the corresponding ".m" file with the code is self-contained and no additional path variables have to be set up in MATLAB. The code is written in such a manner that the ".m" file can be run directly (it loads the necessary data from the corresponding ".mat" file). The code in the "sim\" and "expm\" folders correspond to examples running on simulated and experimental data, respectively.

Estimating an unobserved state x_k from a single binary observation n_k gives rise to the simplest state-space model and EM algorithm equations. The state-space model with only n_k was originally developed in [4]. The code for running the examples for this model are in the "one_bin\sim" and "one_bin\expm" folders. The "one_bin\sim" folder contains the "filter_one_bin.m" and the "data_one_bin.mat" files. The ".m" file contains the code and the ".mat" file contains the data. We will use a similar naming style for all the code examples accompanying this book.

The state-space model we considered in this chapter contained the term β_0 in p_k. However, we did not yet explain how it was calculated. In several studies involving behavioral learning experiments (e.g., [4]), β_0 was determined empirically instead of being estimated as a separate term at the parameter estimation step. Now

$$p_k = \frac{1}{1 + e^{-(\beta_0 + x_k)}} \implies \log\left(\frac{p_k}{1 - p_k}\right) = \beta_0 + x_k, \tag{3.68}$$

and if we assume that $x_k \approx 0$ at the very beginning, we have

$$\beta_0 \approx \log\left(\frac{p_0}{1 - p_0}\right). \tag{3.69}$$

We can use this to calculate β_0 [4]. But what is p_0? In a typical learning experiment involving correct/incorrect responses, p_0 can be taken to be the probability of getting an answer correct prior to any learning taking place. For instance, if there are only two possible answers in each trial, then $p_0 = 0.5$. If there are four possible answers from which to choose, $p_0 = 0.25$. Similarly, in experiments involving the estimation of sympathetic arousal from skin conductance, p_0 can be taken to be the person's

baseline probability of neural impulse occurrence. If the experiment involves both relaxation and stress periods, this baseline can be approximated by the average probability of an impulse occurring in the whole data.

Let us first consider a basic outline of the code itself. The code takes the binary inputs n_k for which we use the variable `n`. Only a few parameters need to be set in this particular code. One of the parameters is the baseline probability p_0 for which we use the variable `base_prob`. In general, we will set `base_prob` to the average probability of $n_k = 1$ occurring in the data. Recall that in the EM algorithm, we repeat the state estimation step and the parameter estimation step until the model parameters converge. In this code example, we use the variable `tol` to determine the tolerance level. Here we have set it to 10^{-6} (i.e., the EM algorithm continues to execute until there is no change in the model parameters to a precision level in the order of 10^{-6}). The variable `ve` denotes the process noise variance. We also use `x_pred`, `x_updt`, and `x_smth` to denote $x_{k|k-1}$, $x_{k|k}$, and $x_{k|K}$, respectively. We also use `v_pred`, `v_updt`, and `v_smth` to denote the corresponding variances $\sigma^2_{k|k-1}$, $\sigma^2_{k|k}$, and $\sigma^2_{k|K}$. Prior to performing all the computations, the model parameters need to be initialized at some values. Here we have initialized the process noise variance to 0.005 and set the initial value of the x_k to 0.

```
base_prob = sum(n) / length(n);
tol = 1e-6; % convergence criteria

ve(1) = 0.005;
x_smth(1) = 0;
b0 = log(base_prob / (1 - base_prob));
```

At a given iteration of the EM algorithm, the code first proceeds in the forward direction from $k = 1, 2, \ldots, K$ calculating both $x_{k|k-1}$ and $x_{k|k}$.

```
x_pred(k) = x_updt(k - 1);
v_pred(k) = v_updt(k - 1) + ve(m);

x_updt(k) = get_state_update(x_pred(k), v_pred(k), b0, n(k));
p_updt(k) = 1 / (1 + exp((-1) * (b0 + x_updt(k))));
v_updt(k) = 1 / ((1 / v_pred(k)) + p_updt(k) * (1 - p_updt(k)));
```

Here the mean state update $x_{k|k}$ is calculated using the function shown below (Newton–Raphson method).

```
function [y] = get_state_update(x_pred, v_pred, b0, n)

    M = 50;    % maximum iterations

    it = zeros(1, M);
    func = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)
       func(i) = it(i) - x_pred - v_pred * (n - exp(b0 + it(i)) /
     (1 + exp(b0 + it(i))));
```

```
        df(i) = 1 + v_pred * exp(b0 + it(i)) / ((1 + exp(b0 + it(i
    ))) ^ 2);
        it(i + 1)  = it(i) - func(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
            y = it(i + 1);
           return
        end
    end

    error('Newton-Raphson failed to converge.');

end
```

After proceeding in the forward direction, we reverse direction and proceed through $k = K, (K-1), \ldots, 1$ to obtain the smoothened $x_{k|K}$ and $\sigma^2_{k|K}$ values. In the code shown below, the variables `W` and `CW` denote U_k and $U_{k,k+1}$ in (3.53) and (3.54), respectively.

```
x_smth(K) = x_updt(K);
v_smth(K) = v_updt(K);
W(K) = v_smth(K) + (x_smth(K) ^ 2);

A(1:(end - 1)) = v_updt(1:(end - 1)) ./ v_pred(2:end);

for k = (K - 1):(-1):1
  x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k + 1));
  v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) - v_pred(k
    + 1));

  CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
  W(k) = v_smth(k) + (x_smth(k) ^ 2);
end
```

After performing state estimation at a particular iteration, we then perform parameter estimation. The state estimation and the parameter estimation steps continue to be executed in turn until convergence.

3.5.1 *Application to Skin Conductance and Sympathetic Arousal*

Running both the simulated and experimental data examples produces the results shown in Fig. 3.3. The code running on simulated data implements the EM algorithm described in this chapter. The code running on experimental data, on the other hand, runs a slightly modified version closer to what was implemented in [25, 26] for estimating sympathetic arousal based on skin conductance. This version of the code additionally attempts to estimate the starting state x_0 as a separate model parameter.

If this code is used to estimate sympathetic arousal based on skin conductance, the only input that is required is the sequence of n_k values (denoted by the variable

n) that represents the presence or absence of neural impulses responsible for SCRs. Ideally, the sequence of neural impulses must be extracted by deconvolving the skin conductance data using a deconvolution procedure such as described in [47]. If, however, deconvolution of the skin conductance data is not possible, a simpler peak detection mechanism could be also used to provide these locations (peak detection was used in [25] and deconvolution was used in [26] for sympathetic arousal estimation). Also with the experimental data, and in several other examples that follow, we use the term "HAI" to denote "High Arousal Index" since many of our examples involve the estimation of sympathetic *arousal* from physiological data. The HAI is inspired by the "Ideal Observer Certainty" term in [4] and is an estimate of how much p_k is above a certain baseline. The HAI can also be calculated based on x_k exceeding an equivalent baseline since p_k is related to x_k.

The right sub-figure in Fig. 3.3 provides an example of how sympathetic arousal varied for a particular subject engaged in an experiment involving different stressors. The experiment is described in [53]. The first three shaded backgrounds correspond

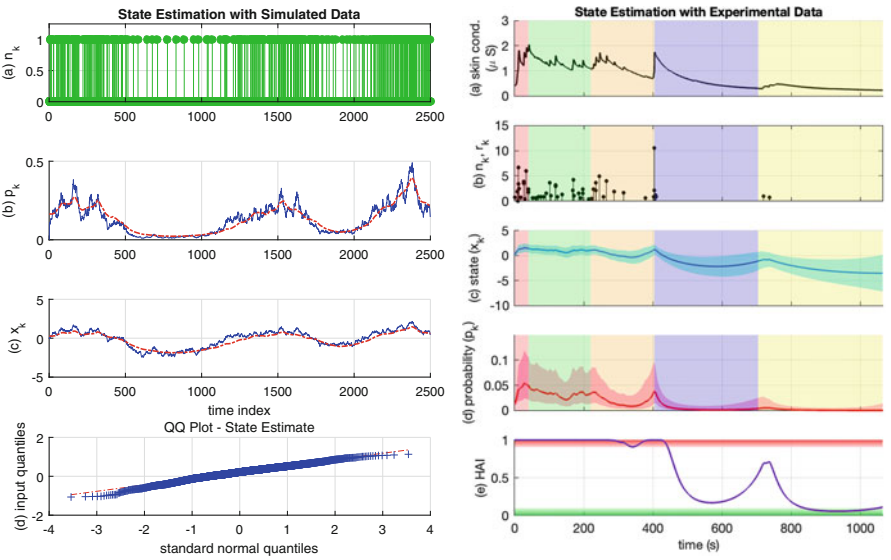

Fig. 3.3 State estimation based on observing one binary variable. The left sub-figure depicts estimation on simulated data, and the right sub-figure depicts the estimation of sympathetic arousal from skin conductance data. The sub-panels on the left, respectively, depict: (**a**) the binary event occurrences n_k; (**b**) the probability of binary event occurrence p_k (blue) and its estimate (red); (**c**) the state x_k (blue) and its estimate (red); (**d**) the quantile–quantile (QQ) plot for the residual error of x_k. The sub-panels on the right, respectively, depict: (**a**) the skin conductance signal; (**b**) the neural impulses; (**c**) the arousal state x_k and its 95% confidence limits; (**d**) the probability of impulse occurrence and its 95% confidence limits; (**e**) the HAI (the regions above 90% and below 10% are shaded in red and green, respectively). The background colors on the right sub-figure correspond to the instruction period, a counting task, a color–word association task, relaxation, and watching a horror movie clip. From [32], used under Creative Commons CC-BY license

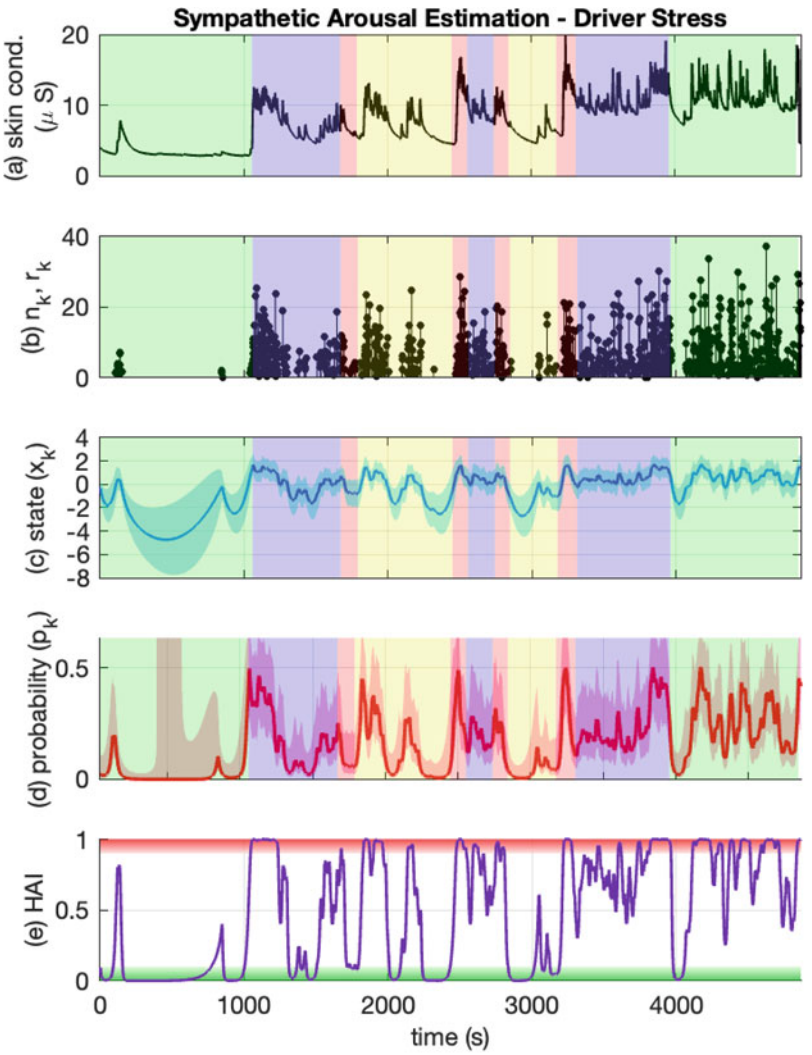

Fig. 3.4 Driver stress estimation. The sub-panels, respectively, depict: (**a**) the skin conductance signal; (**b**) the neural impulses; (**c**) the arousal state x_k and its 95% confidence limits; (**d**) the probability of impulse occurrence and its 95% confidence limits; (**e**) the HAI (the regions above 90% and below 10% are shaded in red and green, respectively). The background colors in turn denote rest, city driving, toll road, highway, toll road, city driving, toll road, highway, toll road, city driving, and rest. From [32], used under Creative Commons CC-BY license

to a period of instructions followed by two cognitive tasks. Arousal remains high during this period. Arousal drops significantly during the relaxation period that follows and briefly increases at the beginning of the emotional stressor (horror movie) after that. Figure 3.4 also provides an additional example of how arousal

varied in a driver stress experiment. The data come from the study described in [54]. In the experiment, each subject had to drive a vehicle along a set route comprising of city driving, toll roads, and highways. Figure 3.4 shows how sympathetic arousal varied during the different road conditions and the rest periods that preceded and followed the actual drive.

Chapter 4
State-Space Model with One Binary and One Continuous Observation

In this chapter, we will consider the case where x_k evolves with time following a slightly more complicated state equation and gives rise to both a binary observation n_k and a continuous observation r_k. Prior to looking into the equation derivations, however, as in the previous chapter, we will again first consider a few example scenarios where the need for such a model arises.

In the previous chapter, we considered the estimation of a continuous-valued learning state x_k based on correct/incorrect responses in a sequence of experimental trials. Based on a state-space model consisting of x_k and the binary observations n_k, the cognitive learning state of an animal could be estimated over time [4]. Note, however, that it is not just the correct/incorrect responses that contain information regarding the animal's learning state. How *fast* the animal responds also reflects changes in learning. For instance, as an animal gradually begins to learn to recognize a specific visual target, not only do the correct answers begin to occur more frequently, but the time taken to respond in each of the trials also starts decreasing (Fig. 4.1). Thus, a state-space model with both a binary observation n_k and a continuous observation r_k was developed in [5] to estimate learning. This was an improvement over the earlier model in [4].

This particular state-space model is not just limited to cognitive learning. It can also be adapted to other applications as well. Human emotion is typically accounted for along two different axes known as valence and arousal [55]. Valence denotes the pleasant–unpleasant nature of an emotion, while arousal denotes its corresponding activation or excitement. Emotional arousal is closely tied to the activation of the sympathetic nervous system [56, 57]. Changes in arousal can occur regardless of the valence of the emotion (i.e., arousal can be high when the emotion is negative,

Supplementary Information The online version contains supplementary material available at (https://doi.org/10.1007/978-3-031-47104-9_4).

GitHub https://github.com/computational-medicine-lab/A-Tutorial-on-the-Design-of-Bayesian-Filters/tree/main/one_bin_one_cont

D. S. Wickramasuriya, R. T. Faghih, *Bayesian Filter Design for Computational Medicine*, https://doi.org/10.1007/978-3-031-47104-9_4

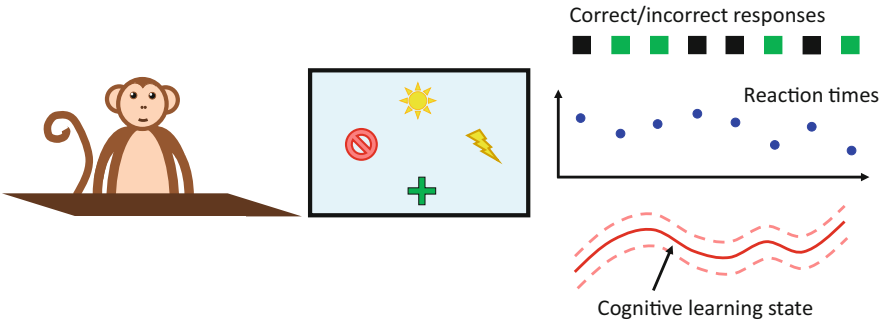

Fig. 4.1 A monkey in a learning experiment with binary-valued correct/incorrect responses where reaction times are recorded. In a behavioral learning experiment, not only do the binary-valued correct/incorrect responses contain information regarding learning, but the time taken to respond in each trial also reflects changes in learning. The state-space model with correct/incorrect responses and reaction times was used in [5] to estimate a cognitive learning state in animal models

as in the case of rage, or when it is positive, as in the case of excitement). As we saw in the earlier chapter, skin conductance is a sensitive index of arousal. Changes in emotional valence, on the other hand, often cause changes in facial expressions. Information regarding these facial expressions can be captured via EMG sensors attached to the face. The state-space model with one binary observation n_k and one continuous observation r_k was used in [27] for an emotional valence recognition application based on EMG signals. In [27], Yadav et al. extracted both a binary feature and a continuous feature based on EMG amplitudes and powers from data in an experiment where subjects watched a series of music videos meant to evoke different emotions. Based on the model, they were able to extract a continuous-valued emotional valence state x_k over time. The same model was also used in [28] for detecting epileptic seizures. Here, the authors extracted a binary feature and a continuous feature from scalp EEG signals to detect the occurrence of epileptic seizures. Based on the features, a continuous-valued seizure severity state could be tracked over time. These examples serve to illustrate the possibility of using physiological state-space models for a wide variety of applications.

4.1 Deriving the Predict Equations in the State Estimation Step

Let us now consider the state-space model itself. Assume that x_k varies with time as

$$x_k = \rho x_{k-1} + \varepsilon_k, \tag{4.1}$$

where ρ is a constant (forgetting factor) and $\varepsilon_k \sim \mathcal{N}(0, \sigma_\varepsilon^2)$. As in the previous chapter, we will, for the time-being, ignore that this is part of state-space control system, and instead view the equation purely in terms of a relationship between

three random variables. As before, we will also consider the derivation of the mean and variance of x_k using basic formulas. We first consider the mean.

$$\mathbb{E}[x_k] = \mathbb{E}[\rho x_{k-1} + \varepsilon_k] \tag{4.2}$$
$$= \mathbb{E}[\rho x_{k-1}] + \mathbb{E}[\varepsilon_k] \text{ using (2.1)} \tag{4.3}$$
$$= \rho\mathbb{E}[x_{k-1}] + \mathbb{E}[\varepsilon_k] \text{ using (2.3)} \tag{4.4}$$
$$= \rho\mathbb{E}[x_{k-1}] \text{ since } \mathbb{E}[\varepsilon_k] = 0 \tag{4.5}$$
$$\therefore \mathbb{E}[x_k] = \rho x_{k-1|k-1}. \tag{4.6}$$

Next we consider the variance.

$$V(x_k) = V(\rho x_{k-1} + \varepsilon_k) \tag{4.7}$$
$$= V(\rho x_{k-1}) + V(\varepsilon_k) + 2Cov(\rho x_{k-1}, \varepsilon_k) \text{ using (2.4)} \tag{4.8}$$
$$= V(\rho x_{k-1}) + V(\varepsilon_k) \text{ since } \varepsilon_k \text{ is uncorrelated with any of the } x_k \text{ terms} \tag{4.9}$$
$$= \rho^2 V(x_{k-1}) + V(\varepsilon_k) \text{ using (2.6)} \tag{4.10}$$
$$\therefore V(x_k) = \rho^2\sigma^2_{k-1|k-1} + \sigma^2_\varepsilon. \tag{4.11}$$

Now that we know the mean and variance of x_k, we can use the fact that it is also Gaussian distributed to state that

$$p(x_k|n_{1:k-1}, r_{1:k-1}) = \frac{1}{\sqrt{2\pi\sigma^2_{k|k-1}}} e^{\frac{-(x_k - x_{k|k-1})^2}{2\sigma^2_{k|k-1}}}. \tag{4.12}$$

When x_k evolves with time following $x_k = \rho x_{k-1} + \varepsilon_k$, the predict equations in the state estimation step are

$$x_{k|k-1} = \rho x_{k-1|k-1} \tag{4.13}$$
$$\sigma^2_{k|k-1} = \rho^2\sigma^2_{k-1|k-1} + \sigma^2_\varepsilon. \tag{4.14}$$

4.2 Deriving the Update Equations in the State Estimation Step

In the current model, x_k gives rise to a continuous-valued observation r_k in addition to n_k. We shall assume that x_k is related to r_k through a linear relationship.

$$r_k = \gamma_0 + \gamma_1 x_k + v_k, \tag{4.15}$$

where γ_0 and γ_1 are constants and $v_k \sim \mathcal{N}(0, \sigma_k^2)$ is sensor noise. Our sensor readings y_k now consist of both r_k and n_k. What would be the best estimate of x_k once we have observed y_k? Just like in the previous chapter, we will make use of the result in (2.16) to derive this estimate. First, however, we need to note that

$$p(r_k|x_k) = \frac{1}{\sqrt{2\pi\sigma_v^2}} e^{\frac{-(r_k-\gamma_0-\gamma_1 x_k)^2}{2\sigma_v^2}}. \tag{4.16}$$

This can be easily verified from (4.15). We are now ready to derive the best estimate (mean) for x_k and its uncertainty. We will need to make use of $p(r_k|x_k)$, $p(n_k|x_k)$, and $p(x_k|n_{1:k-1}, r_{1:k-1})$ to derive this estimate. Note that we now have an additional exponent term for r_k in $p(x_k|n_{1:k}, r_{1:k})$. Using (2.16), we have

$$\begin{aligned} &p(x_k|n_{1:k}, r_{1:k}) \\ &= \frac{p(n_k|x_k, n_{1:k-1}, r_{1:k-1})\, p(r_k|x_k, n_{1:k-1}, r_{1:k-1})\, p(x_k|n_{1:k-1}, r_{1:k-1})}{p(n_k, r_k|n_{1:k-1}, r_{1:k-1})} \end{aligned} \tag{4.17}$$

$$\propto p(n_k|x_k)\, p(r_k|x_k)\, p(x_k|n_{1:k-1}, r_{1:k-1}) \tag{4.18}$$

$$\propto e^{n_k \log(p_k) + (1-n_k)\log(1-p_k)} \times e^{\frac{-(r_k-\gamma_0-\gamma_1 x_k)^2}{2\sigma_v^2}} \times e^{\frac{-(x_k - x_{k|k-1})^2}{2\sigma_{k|k-1}^2}}. \tag{4.19}$$

Taking the log on both sides, we have

$$\begin{aligned} q = {} & n_k \log(p_k) + (1-n_k)\log(1-p_k) - \frac{(r_k-\gamma_0-\gamma_1 x_k)^2}{2\sigma_v^2} \\ & - \frac{(x_k - x_{k|k-1})^2}{2\sigma_{k|k-1}^2} + \text{constant}. \end{aligned} \tag{4.20}$$

The mean and variance of x_k can now be derived by taking the first and second derivatives of q. Making use of (3.39), we have

$$\frac{dq}{dx_k} = n_k - p_k + \frac{\gamma_1(r_k - \gamma_0 - \gamma_1 x_k)}{\sigma_v^2} - \frac{(x_k - x_{k|k-1})}{\sigma_{k|k-1}^2} = 0. \tag{4.21}$$

We will use a small trick to solve for x_k in the equation above. We will add and subtract the term $\gamma_1 x_{k|k-1}$ in the term involving r_k.

$$n_k - p_k + \frac{\gamma_1(r_k - \gamma_0 - \gamma_1 x_k + \gamma_1 x_{k|k-1} - \gamma_1 x_{k|k-1})}{\sigma_v^2} = \frac{(x_k - x_{k|k-1})}{\sigma_{k|k-1}^2} \tag{4.22}$$

$$n_k - p_k + \frac{\gamma_1(r_k - \gamma_0 - \gamma_1 x_{k|k-1})}{\sigma_v^2} - \frac{\gamma_1^2}{\sigma_v^2}(x_k - x_{k|k-1}) = \frac{(x_k - x_{k|k-1})}{\sigma_{k|k-1}^2} \tag{4.23}$$

$$n_k - p_k + \frac{\gamma_1(r_k - \gamma_0 - \gamma_1 x_{k|k-1})}{\sigma_v^2} = (x_k - x_{k|k-1})\left(\frac{1}{\sigma_{k|k-1}^2} + \frac{\gamma_1^2}{\sigma_v^2}\right) \tag{4.24}$$

$$\frac{\sigma_v^2(n_k - p_k) + \gamma_1(r_k - \gamma_0 - \gamma_1 x_{k|k-1})}{\sigma_v^2} = (x_k - x_{k|k-1})\left(\frac{\sigma_v^2 + \gamma_1^2\sigma_{k|k-1}^2}{\sigma_{k|k-1}^2\sigma_v^2}\right). \tag{4.25}$$

This yields the mean update

$$x_k = x_{k|k-1} + \frac{\sigma_{k|k-1}^2}{\gamma_1^2\sigma_{k|k-1}^2 + \sigma_v^2}\Big[\sigma_v^2(n_k - p_k) + \gamma_1(r_k - \gamma_0 - \gamma_1 x_{k|k-1})\Big]. \tag{4.26}$$

Again, to clarify the explicit dependence of p_k on x_k and the fact that this is the estimate of x_k having observed $n_{1:k}$ and $r_{1:k}$ (the sensor readings up to time index k), we shall say

$$x_{k|k} = x_{k|k-1} + \frac{\sigma_{k|k-1}^2}{\gamma_1^2\sigma_{k|k-1}^2 + \sigma_v^2}\Big[\sigma_v^2(n_k - p_{k|k}) + \gamma_1(r_k - \gamma_0 - \gamma_1 x_{k|k-1})\Big]. \tag{4.27}$$

We next take the second derivative of q similar to (3.40). This yields

$$\frac{d^2q}{dx_k^2} = -p_k(1 - p_k) - \frac{\gamma_1^2}{\sigma_v^2} - \frac{1}{\sigma_{k|k-1}^2}. \tag{4.28}$$

Based on (2.21), the uncertainty or variance associated with the new state estimate $x_{k|k}$, therefore, is

$$\sigma_{k|k}^2 = -\left(\frac{d^2q}{dx_k^2}\right)^{-1} = \left[\frac{1}{\sigma_{k|k-1}^2} + p_{k|k}(1 - p_{k|k}) + \frac{\gamma_1^2}{\sigma_v^2}\right]^{-1}. \tag{4.29}$$

When x_k gives rise to a binary observation n_k and a continuous observation r_k, the update equations in the state estimation step are

$$x_{k|k} = x_{k|k-1} + \frac{\sigma^2_{k|k-1}}{\gamma_1^2\sigma^2_{k|k-1} + \sigma_v^2}\Big[\sigma_v^2(n_k - p_{k|k}) + \gamma_1(r_k - \gamma_0 - \gamma_1 x_{k|k-1})\Big] \tag{4.30}$$

$$\sigma^2_{k|k} = \left[\frac{1}{\sigma^2_{k|k-1}} + p_{k|k}(1 - p_{k|k}) + \frac{\gamma_1^2}{\sigma_v^2}\right]^{-1}. \tag{4.31}$$

4.3 Deriving the Parameter Estimation Step Equations

In the previous chapter, we only needed to derive the update equation for the process noise variance σ_ε^2 at the parameter estimation step. In the current model, we have a few more parameters. Thus we will need to derive the update equations for ρ, γ_0, γ_1, and σ_v^2 in addition to the update for σ_ε^2.

4.3.1 Deriving the Process Noise Variance

The derivation of the process noise variance update is very similar to the earlier case in the preceding chapter. In fact, the only difference from (3.62) is that we will now have ρx_{k-1} in the log-likelihood term instead of x_{k-1}. We shall label the required log-likelihood term Q_1.

$$Q_1 = \frac{-K}{2}\log\left(2\pi\sigma_\varepsilon^2\right) - \sum_{k=2}^{K}\frac{\mathbb{E}\Big[(x_k - \rho x_{k-1})^2\Big]}{2\sigma_\varepsilon^2}. \tag{4.32}$$

We take the partial derivative of Q_1 with respect to σ_ε^2 and set it to 0 to solve for the parameter estimation step update.

$$\frac{\partial Q_1}{\partial\sigma_\varepsilon^2} = \frac{-K}{2\sigma_\varepsilon^2} + \frac{1}{2\sigma_\varepsilon^4}\sum_{k=2}^{K}\mathbb{E}\Big[(x_k - \rho x_{k-1}\Big] = 0 \tag{4.33}$$

$$\implies \sigma_\varepsilon^2 = \frac{1}{K}\sum_{k=2}^{K}\Big\{\mathbb{E}\big[x_k^2\big] - 2\rho\mathbb{E}\big[x_k x_{k-1}\big] + \rho^2\mathbb{E}\big[x_{k-1}^2\big]\Big\} \tag{4.34}$$

$$= \frac{1}{K}\left\{ \sum_{k=2}^{K} U_k - 2\rho \sum_{k=1}^{K-1} U_{k,k+1} + \rho^2 \sum_{k=1}^{K-1} U_k \right\}. \tag{4.35}$$

The parameter estimation step update for σ_ε^2 when x_k evolves with time following $x_k = \rho x_{k-1} + \varepsilon_k$ is

$$\sigma_\varepsilon^2 = \frac{1}{K}\left\{ \sum_{k=2}^{K} U_k - 2\rho \sum_{k=1}^{K-1} U_{k,k+1} + \rho^2 \sum_{k=1}^{K-1} U_k \right\}. \tag{4.36}$$

4.3.2 Deriving the Forgetting Factor

We will take the partial derivative of Q_1 in (4.32) with respect to ρ and set it to 0 to solve for its parameter estimation step update.

$$\frac{\partial Q_1}{\partial \rho} = \frac{(-1)}{2\sigma_\varepsilon^2} \sum_{k=2}^{K} \mathbb{E}\big[-2x_{k-1}(x_k - \rho x_{k-1}) \big] \tag{4.37}$$

$$\implies 0 = -\sum_{k=2}^{K} \mathbb{E}\big[x_k x_{k-1}\big] + \rho \sum_{k=2}^{K} \mathbb{E}\big[x_{k-1}^2\big] \tag{4.38}$$

$$= -\sum_{k=1}^{K-1} U_{k,k+1} + \rho \sum_{k=1}^{K-1} U_k \tag{4.39}$$

$$\rho = \sum_{k=1}^{K-1} U_{k,k+1} \left[\sum_{k=1}^{K-1} U_k \right]^{-1}. \tag{4.40}$$

The parameter estimation step update for ρ when x_k evolves with time following $x_k = \rho x_{k-1} + \varepsilon_k$ is

$$\rho = \sum_{k=1}^{K-1} U_{k,k+1} \left[\sum_{k=1}^{K-1} U_k \right]^{-1}. \tag{4.41}$$

4.3.3 Deriving the Constant Coefficient Terms

We will next consider the model parameters that are related to r_k. Recall from (3.57) that we need to maximize the expected value of the log of the joint probability

$$p(x_{1:K} \cap y_{1:K}|\Theta) = p(y_{1:K}|x_{1:K}, \Theta)p(x_{1:K}|\Theta). \tag{4.42}$$

In the current state-space model, y_k comprises both n_k and r_k. The probability term containing γ_0, γ_1, and σ_v^2 is

$$p(r_{1:K}|x_{1:K}, \Theta) = \prod_{k=1}^{K} \frac{1}{\sqrt{2\pi\sigma_v^2}} e^{\frac{-(r_k-\gamma_0-\gamma_1 x_k)^2}{2\sigma_v^2}}. \tag{4.43}$$

Let us first take the log of this term followed by the expected value. Labeling this as Q_2, we have

$$Q_2 = \frac{-K}{2}\log\left(2\pi\sigma_v^2\right) - \sum_{k=1}^{K} \frac{\mathbb{E}\left[(r_k - \gamma_0 - \gamma_1 x_k)^2\right]}{2\sigma_v^2}. \tag{4.44}$$

To solve for γ_0 and γ_1, we have to take the partial derivatives of Q_2 with respect to γ_0 and γ_1, set them each to 0, and solve the resulting equations. We first take the partial derivative with respect to γ_0.

$$\frac{\partial Q_2}{\partial \gamma_0} = \frac{1}{2\sigma_v^2}\sum_{k=1}^{K} 2\mathbb{E}\left[r_k - \gamma_0 - \gamma_1 x_k\right] = 0 \tag{4.45}$$

$$\implies 0 = \sum_{k=1}^{K} r_k - \gamma_0 K - \gamma_1 \sum_{k=1}^{K} \mathbb{E}\left[x_k\right] \tag{4.46}$$

$$= \sum_{k=1}^{K} r_k - \gamma_0 K - \gamma_1 \sum_{k=1}^{K} x_{k|K} \tag{4.47}$$

$$\gamma_0 K + \gamma_1 \sum_{k=1}^{K} x_{k|K} = \sum_{k=1}^{K} r_k. \tag{4.48}$$

This provides one equation containing the two unknowns γ_0 and γ_1. We next take the partial derivative with respect to γ_1.

$$\frac{\partial Q_2}{\partial \gamma_1} = \frac{1}{2\sigma_v^2}\sum_{k=1}^{K} 2\mathbb{E}\left[x_k(r_k - \gamma_0 - \gamma_1 x_k)\right] = 0 \tag{4.49}$$

$$\begin{aligned}\implies 0 &= \sum_{k=1}^{K} r_k \mathbb{E}[x_k] - \gamma_0 \sum_{k=1}^{K} \mathbb{E}[x_k] - \gamma_1 \sum_{k=1}^{K} \mathbb{E}[x_k^2] \\ &= \sum_{k=1}^{K} r_k x_{k|K} - \gamma_0 \sum_{k=1}^{K} x_{k|K} - \gamma_1 \sum_{k=1}^{K} U_k \end{aligned} \tag{4.50}$$

$$\gamma_0 \sum_{k=1}^{K} x_{k|K} + \gamma_1 \sum_{k=1}^{K} U_k = \sum_{k=1}^{K} r_k x_{k|K}. \tag{4.51}$$

This provides the second equation necessary to solve for γ_0 and γ_1.

The parameter estimation step updates for γ_0 and γ_1 when we observe a continuous variable $r_k = \gamma_0 + \gamma_1 x_k + v_k$ are

$$\begin{bmatrix} \gamma_0 \\ \gamma_1 \end{bmatrix} = \begin{bmatrix} K & \sum_{k=1}^{K} x_{k|K} \\ \sum_{k=1}^{K} x_{k|K} & \sum_{k=1}^{K} U_k \end{bmatrix}^{-1} \begin{bmatrix} \sum_{k=1}^{K} r_k \\ \sum_{k=1}^{K} r_k x_{k|K} \end{bmatrix}. \tag{4.52}$$

4.3.4 *Deriving the Sensor Noise Variance*

The term Q_2 in (4.44) also contains the sensor noise variance σ_v^2.

$$Q_2 = \frac{-K}{2} \log\left(2\pi\sigma_v^2\right) - \sum_{k=1}^{K} \frac{\mathbb{E}\left[(r_k - \gamma_0 - \gamma_1 x_k)^2\right]}{2\sigma_v^2}. \tag{4.53}$$

We take its partial derivative with respect to σ_v^2 and set it to 0 to solve for σ_v^2.

$$\frac{\partial Q_2}{\partial \sigma_v^2} = \frac{-K}{2\sigma_v^2} + \frac{1}{2\sigma_v^4} \sum_{k=1}^{K} \mathbb{E}\left[(r_k - \gamma_0 - \gamma_1 x_k)^2\right] = 0 \tag{4.54}$$

$$\begin{aligned}\implies \sigma_v^2 &= \frac{1}{K} \sum_{k=1}^{K} \mathbb{E}\left[(r_k - \gamma_0 - \gamma_1 x_k)^2\right] \\ &= \frac{1}{K} \left\{ \sum_{k=1}^{K} r_k^2 + K\gamma_0^2 + \gamma_1^2 \sum_{k=1}^{K} \mathbb{E}[x_k^2] - 2\gamma_0 \sum_{k=1}^{K} r_k - 2\gamma_1 \sum_{k=1}^{K} r_k \mathbb{E}[x_k] \right.\end{aligned}$$

$$+ 2\gamma_0\gamma_1 \sum_{k=1}^{K} \mathbb{E}[x_k] \Bigg\}$$

$$= \frac{1}{K} \Bigg\{ \sum_{k=1}^{K} r_k^2 + K\gamma_0^2 + \gamma_1^2 \sum_{k=1}^{K} U_k - 2\gamma_0 \sum_{k=1}^{K} r_k - 2\gamma_1 \sum_{k=1}^{K} r_k x_{k|K}$$

$$+ 2\gamma_0\gamma_1 \sum_{k=1}^{K} x_{k|K} \Bigg\}. \tag{4.55}$$

The parameter estimation step update for σ_v^2 when we observe a continuous variable $r_k = \gamma_0 + \gamma_1 x_k + v_k$ is

$$\sigma_v^2 = \frac{1}{K} \Bigg\{ \sum_{k=1}^{K} r_k^2 + K\gamma_0^2 + \gamma_1^2 \sum_{k=1}^{K} U_k - 2\gamma_0 \sum_{k=1}^{K} r_k - 2\gamma_1 \sum_{k=1}^{K} r_k x_{k|K}$$

$$+ 2\gamma_0\gamma_1 \sum_{k=1}^{K} x_{k|K} \Bigg\}. \tag{4.56}$$

4.4 MATLAB Examples

The MATLAB code examples for implementing the EM algorithm described in this chapter are provided in the following folders:

- one_bin_one_cont\

  sim\

  data_one_bin_one_cont.mat
  filter_one_bin_one_cont.m

  expm\

  expm_data_one_bin_two_cont.mat
  expm_filter_one_bin_one_cont.m

Note that the code implements a slightly different version of what was discussed here in that the state equation does not contain ρ. Code examples containing ρ and αI_k are provided in the following chapter for the case where one binary and two continuous observations are present in the state-space model. The current code can easily be modified if ρ is to be included.

The code for this particular state-space model is an extension of the earlier model. It takes in as input variables `n` and `r` that denote n_k and r_k, respectively. We use `r0`, `r1`, and `vr` for γ_0, γ_1, and σ_v^2. Shown below is a part of the code where β_0 is calculated and the model parameters are initialized.

```
base_prob = sum(n) / length(n);
tol = 1e-8; % convergence criteria

ve(1) = 0.005;
x_smth(1) = 0;
r0(1) = 0.1;
r1(1) = r(1);
vr(1) = 0.002;
b0 = log(base_prob / (1 - base_prob));
```

Similar to the code examples in the preceding chapter, we also use `x_pred`, `x_updt`, and `x_smth` to denote $x_{k|k-1}$, $x_{k|k}$, and $x_{k|K}$, respectively. Similarly, `v_pred`, `v_updt`, and `v_smth` are used to denote the corresponding variances $\sigma^2_{k|k-1}$, $\sigma^2_{k|k}$, and $\sigma^2_{k|K}$. Just like in the earlier case as well, the code first progresses through the time indices $k = 1, 2, \ldots, K$ at the state estimation step.

```
x_pred(k) = x_updt(k - 1);
v_pred(k) = v_updt(k - 1) + ve(m);

x_updt(k) = get_posterior_mode(x_pred(k), v_pred(k), r(k), r0(m),
     r1(m), vr(m), b0, n(k));
p_updt(k) = 1 / (1 + exp((-1) * (b0 + x_updt(k))));
v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(m)) + p_updt
    (k) * (1 - p_updt(k)));
```

The update for $x_{k|k}$ is calculated using the function shown below based on both n_k and r_k.

```
function [y] = get_posterior_mode(x_pred, v_pred, z, r0, r1, vr,
    b0, n)

    M = 100;    % maximum iterations

    it = zeros(1, M);
    f = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)
        C = v_pred / ((r1 ^ 2) * v_pred + vr);
        f(i) = it(i) - x_pred - C * (r1 * (z - r0 - r1 * x_pred)
    + vr * (n - (1 / (1 + exp((-1) * (b0 + it(i)))))));
        df(i) = 1 + C * vr * exp(b0 + it(i)) / ((1 + exp(b0 + it(
    i))) ^ 2);

        it(i + 1)  = it(i) - f(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
```

```
            y = it(i + 1);
            return
        end
    end

    error('Newton-Raphson failed to converge.');

end
```

The smoothing step also remains the same (there would have been a change if ρ was included).

```
x_smth(K) = x_updt(K);
v_smth(K) = v_updt(K);
W(K) = v_smth(K) + (x_smth(K) ^ 2);

A(1:(end - 1)) = v_updt(1:(end - 1)) ./ v_pred(2:end);

for k = (K - 1):(-1):1
  x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k + 1));
  v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) - v_pred(k
    + 1));

  CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
  W(k) = v_smth(k) + (x_smth(k) ^ 2);
end
```

The updates for σ_ε^2, γ_0, γ_1, and σ_v^2 are calculated at the parameter estimation step.

4.4.1 Application to EMG and Emotional Valence

Running the simulated and experimental data code examples produces the results shown in Fig. 4.2. The experimental data example relates to emotional valence and EMG. Emotion can be accounted for along two different orthogonal axes known as valence and arousal [55]. Valence refers to the pleasant–unpleasant nature of an emotion. In [27], this state-space model with one binary and one continuous feature was used to estimate emotional valence using EMG signal features. The binary and continuous features were extracted based on the amplitudes and powers of the EMG signal. The data were collected as a part of the study described in [58] where subjects were shown a series of music videos to elicit different emotional responses.

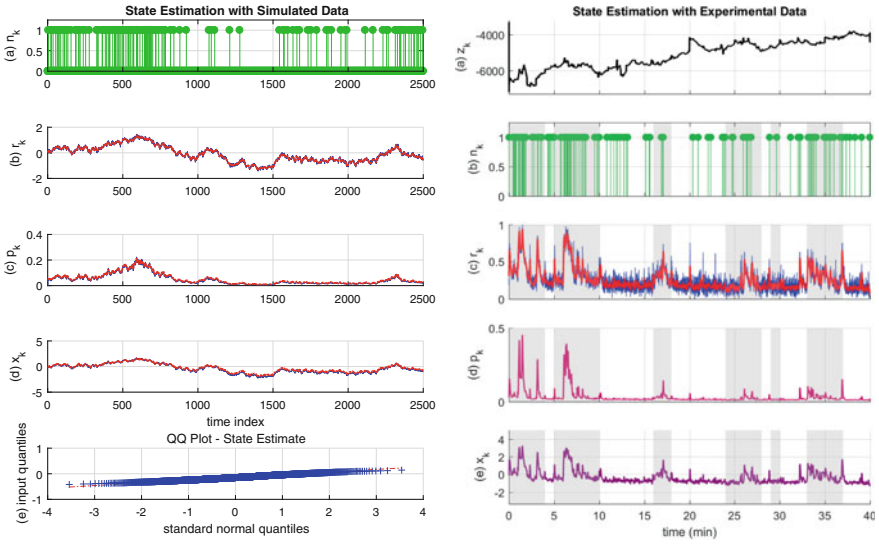

Fig. 4.2 State estimation based on observing one binary and one continuous variable. The left sub-figure depicts estimation on simulated data, and the right sub-figure depicts the estimation of emotional valence from EMG data. The sub-panels on the left, respectively, depict: (**a**) the binary event occurrences n_k; (**b**) the continuous variable r_k (blue) and its estimate (red); (**c**) the probability of binary event occurrence p_k (blue) and its estimate (red); (**d**) the state x_k (blue) and its estimate (red); (**e**) the QQ plot for the residual error of x_k. The sub-panels on the right, respectively, depict: (**a**) the raw EMG signal; (**b**) the binary EMG feature n_k; (**c**) the continuous EMG feature r_k (blue) and its estimate (red); (**d**) the probability of binary event occurrence; (**e**) the emotional valence state x_k. The shaded background colors on the right sub-figure correspond to music videos where subject-provided emotional valence ratings were high

Chapter 5
State-Space Model with One Binary and Two Continuous Observations

In this chapter, we will consider a more complicated form of the state equation—one that contains both a forgetting factor and an external input. We will also extend the earlier state-space model to the case where we now have one binary observation and *two* continuous observations. As before, however, we will first consider a scenario motivating the need for such a state-space model.

Recall the example two chapters ago concerning the estimation of sympathetic arousal from skin conductance features. In reality, it is not just the rate of occurrence of neural impulses to the sweat glands that reflects changes in arousal. Other features in a skin conductance signal also contain arousal information. A skin conductance signal comprises a fast-varying phasic component superimposed on top of a slower-varying tonic component. The phasic component consists of all the SCRs. The amplitudes of these SCRs (or equivalently, the amplitudes of the neural impulses that generated them), in addition to their occurrence, also reflect changes in arousal [59]. In particular, larger SCRs reflect greater sympathetic arousal. Additionally, the tonic level also contains information regarding general arousal [60]. Thus, there are three primary sources of information in a skin conductance signal that capture arousal levels: (i) the occurrence of SCRs (or equivalently the occurrence of the neural impulses that generated the SCRs); (ii) the amplitudes of the SCRs (or the amplitudes of the neural impulses); (iii) the tonic component. These three make up one binary feature and two amplitude (continuous) features. A state-space model based on these three features was developed in [29], for estimating arousal from skin conductance. Here, a transformed version of the SCR amplitudes was interpolated over to derive the first continuous variable, and the tonic component was considered

Supplementary Information The online version contains supplementary material available at (https://doi.org/10.1007/978-3-031-47104-9_5).

GitHub https://github.com/computational-medicine-lab/A-Tutorial-on-the-Design-of-Bayesian-Filters/tree/main/one_bin_two_cont

D. S. Wickramasuriya, R. T. Faghih, *Bayesian Filter Design for Computational Medicine*, https://doi.org/10.1007/978-3-031-47104-9_5

the second continuous variable. Different algorithms are available for separating out the tonic and phasic components in a skin conductance signal (e.g., [61, 62]).

A further note is also worth mentioning here. Recall the $\mathbf{u}_k$ term in our discussion of state-space models at the beginning. Thus far, we have not yet considered a model where an external input also drives the state x_k. In reality, external circumstances and environmental inputs all affect the way we feel. The model in [29] included such an external input term I_k. The model was evaluated on two experimental datasets, one of which involved a Pavlovian fear conditioning experiment. In typical fear conditioning experiments, a neutral cue is paired with an unpleasant stimulus such as a painful electric shock. Through repeated pairing, the neutral cue alone begins to elicit a physiological response that is typically seen for the unpleasant stimulus [63]. In fear conditioning experiments, the unpleasant stimulus could also take other forms such a blast of air to the throat, an aversive image, or a loud sound [64, 65]. In [29], the neutral cues along with the unpleasant shocks were modeled as binary-valued indicator inputs I_k that drove the sympathetic arousal state x_k.

5.1 Deriving the Predict Equations in the State Estimation Step

Let us now turn our attention to the state-space model itself and assume that x_k evolves with time as

$$x_k = \rho x_{k-1} + \alpha I_k + \varepsilon_k, \tag{5.1}$$

where α is a constant and I_k is an external input. The other terms have their usual meanings. Let us again consider how we may derive the mean and variance using basic statistical formulas. Since we know what the external input is, we do not treat it as a random variable but rather as a constant term. We first consider the mean.

$$\begin{aligned}
\mathbb{E}[x_k] &= \mathbb{E}[\rho x_{k-1} + \alpha I_k + \varepsilon_k] && (5.2)\\
&= \mathbb{E}[\rho x_{k-1}] + \mathbb{E}[\alpha I_k] + \mathbb{E}[\varepsilon_k] \text{ using (2.1)} && (5.3)\\
&= \mathbb{E}[\rho x_{k-1}] + \alpha I_k + \mathbb{E}[\varepsilon_k] \text{ using (2.2)} && (5.4)\\
&= \rho\mathbb{E}[x_{k-1}] + \alpha I_k + \mathbb{E}[\varepsilon_k] \text{ using (2.3)} && (5.5)\\
&= \rho\mathbb{E}[x_{k-1}] + \alpha I_k \text{ since } \mathbb{E}[\varepsilon_k] = 0 && (5.6)\\
\therefore \mathbb{E}[x_k] &= \rho x_{k-1|k-1} + \alpha I_k. && (5.7)
\end{aligned}$$

We next consider the variance.

$$\begin{aligned}
V(x_k) &= V(\rho x_{k-1} + \alpha I_k + \varepsilon_k) && (5.8)\\
&= V(\rho x_{k-1} + \alpha I_k) + V(\varepsilon_k) + 2Cov(\rho x_{k-1} + \alpha I_k, \varepsilon_k) \text{ using (2.4)} && (5.9)
\end{aligned}$$

$$= V(\rho x_{k-1} + \alpha I_k) + V(\varepsilon_k)$$

since ε_k is uncorrelated with any of the x_k or I_k terms (5.10)

$$= V(\rho x_{k-1}) + V(\varepsilon_k) \text{ using (2.5)} \tag{5.11}$$

$$= \rho^2 V(x_{k-1}) + V(\varepsilon_k) \text{ using (2.6)} \tag{5.12}$$

$$\therefore V(x_k) = \rho^2 \sigma^2_{k-1|k-1} + \sigma^2_\varepsilon. \tag{5.13}$$

When x_k evolves with time following $x_k = \rho x_{k-1} + \alpha I_k + \varepsilon_k$, the predict equations in the state estimation step are

$$x_{k|k-1} = \rho x_{k-1|k-1} + \alpha I_k \tag{5.14}$$

$$\sigma^2_{k|k-1} = \rho^2 \sigma^2_{k-1|k-1} + \sigma^2_\varepsilon. \tag{5.15}$$

5.2 Deriving the Update Equations in the State Estimation Step

In this state-space model, we include a second continuous variable s_k. Similar to r_k, we will assume that s_k too is linearly related to x_k as

$$s_k = \delta_0 + \delta_1 x_k + w_k, \tag{5.16}$$

where δ_0 and δ_1 are constants and $w_k \sim \mathcal{N}(0, \sigma^2_w)$ is sensor noise. Similar to the case of r_k in (4.16), we also have

$$p(s_k|x_k) = \frac{1}{\sqrt{2\pi\sigma^2_w}} e^{\frac{-(s_k - \delta_0 - \delta_1 x_k)^2}{2\sigma^2_w}}. \tag{5.17}$$

The procedure to derive the update equations in the state estimation step is now similar to what we have seen earlier. With s_k included, we have yet another exponent term in $p(x_k|y_{1:k})$. Therefore,

$$p(x_k|y_{1:k}) \propto p(n_k|x_k)p(r_k|x_k)p(s_k|x_k)p(x_k|n_{1:k-1}, r_{1:k-1}, s_{1:k-1}) \tag{5.18}$$

$$\propto e^{n_k \log(p_k) + (1-n_k)\log(1-p_k)} \times e^{\frac{-(r_k - \gamma_0 - \gamma_1 x_k)^2}{2\sigma^2_v}} \times e^{\frac{-(s_k - \delta_0 - \delta_1 x_k)^2}{2\sigma^2_w}}$$

$$\times e^{\frac{-(x_k - x_{k|k-1})^2}{2\sigma^2_{k|k-1}}}. \tag{5.19}$$

Taking the log on both sides, we have

$$q = n_k \log(p_k) + (1 - n_k) \log(1 - p_k) - \frac{(r_k - \gamma_0 - \gamma_1 x_k)^2}{2\sigma_v^2} - \frac{(s_k - \delta_0 - \delta_1 x_k)^2}{2\sigma_w^2} - \frac{(x_k - x_{k|k-1})^2}{2\sigma_{k|k-1}^2} + \text{constant}. \tag{5.20}$$

Taking the first derivative of q and setting it to 0 yield

$$\frac{dq}{dx_k} = n_k - p_k + \frac{\gamma_1(r_k - \gamma_0 - \gamma_1 x_k)}{\sigma_v^2} + \frac{\delta_1(s_k - \delta_0 - \delta_1 x_k)}{\sigma_w^2} - \frac{(x_k - x_{k|k-1})}{\sigma_{k|k-1}^2} = 0. \tag{5.21}$$

We used a trick in the previous chapter to solve for x_k. We added and subtracted $\gamma_1 x_{k|k-1}$ to the term containing r_k. We will do the same here. We will also add and subtract $\delta_1 x_{k|k-1}$ to the term containing s_k.

$$\frac{(x_k - x_{k|k-1})}{\sigma_{k|k-1}^2} = n_k - p_k + \frac{\gamma_1(r_k - \gamma_0 - \gamma_1 x_k + \gamma_1 x_{k|k-1} - \gamma_1 x_{k|k-1})}{\sigma_v^2} + \frac{\delta_1(s_k - \delta_0 - \delta_1 x_k + \delta_1 x_{k|k-1} - \delta_1 x_{k|k-1})}{\sigma_w^2} \tag{5.22}$$

$$= n_k - p_k + \frac{\gamma_1(r_k - \gamma_0 - \gamma_1 x_{k|k-1})}{\sigma_v^2} + \frac{\delta_1(s_k - \delta_0 - \delta_1 x_{k|k-1})}{\sigma_w^2} - \left(\frac{\gamma_1^2}{\sigma_v^2} + \frac{\delta_1^2}{\sigma_w^2}\right)(x_k - x_{k|k-1}) \tag{5.23}$$

$$= n_k - p_k + \frac{\gamma_1(r_k - \gamma_0 - \gamma_1 x_{k|k-1})}{\sigma_v^2} + \frac{\delta_1(s_k - \delta_0 - \delta_1 x_{k|k-1})}{\sigma_w^2} - \left(\frac{\gamma_1^2\sigma_w^2 + \delta_1^2\sigma_v^2}{\sigma_v^2\sigma_w^2}\right)(x_k - x_{k|k-1}). \tag{5.24}$$

Therefore,

$$\frac{(x_k - x_{k|k-1})}{\sigma_{k|k-1}^2} + \left(\frac{\gamma_1^2\sigma_w^2 + \delta_1^2\sigma_v^2}{\sigma_v^2\sigma_w^2}\right)(x_k - x_{k|k-1}) = n_k - p_k + \frac{\gamma_1(r_k - \gamma_0 - \gamma_1 x_{k|k-1})}{\sigma_v^2} + \frac{\delta_1(s_k - \delta_0 - \delta_1 x_{k|k-1})}{\sigma_w^2} \tag{5.25}$$

$$(x_k - x_{k|k-1})\left[\frac{\sigma_v^2\sigma_w^2 + \sigma_{k|k-1}^2(\gamma_1^2\sigma_w^2 + \delta_1^2\sigma_v^2)}{\sigma_{k|k-1}^2\sigma_v^2\sigma_w^2}\right]$$
$$= \frac{\sigma_v^2\sigma_w^2}{\sigma_v^2\sigma_w^2}(n_k - p_k) + \frac{\gamma_1\sigma_w^2(r_k - \gamma_0 - \gamma_1 x_{k|k-1})}{\sigma_v^2\sigma_w^2}$$
$$+ \frac{\delta_1\sigma_v^2(s_k - \delta_0 - \delta_1 x_{k|k-1})}{\sigma_v^2\sigma_w^2}. \tag{5.26}$$

This yields the state update

$$x_{k|k} = x_{k|k-1} + \frac{\sigma_{k|k-1}^2}{\sigma_v^2\sigma_w^2 + \sigma_{k|k-1}^2(\gamma_1^2\sigma_w^2 + \delta_1^2\sigma_v^2)}$$
$$\times \Big[\sigma_v^2\sigma_w^2(n_k - p_{k|k}) + \gamma_1\sigma_w^2(r_k - \gamma_0 - \gamma_1 x_{k|k-1})$$
$$+ \delta_1\sigma_v^2(s_k - \delta_0 - \delta_1 x_{k|k-1})\Big]. \tag{5.27}$$

Likewise, the second derivative yields

$$\frac{d^2q}{dx_k^2} = -p_k(1 - p_k) - \frac{\gamma_1^2}{\sigma_v^2} - \frac{\delta_1^2}{\sigma_w^2} - \frac{1}{\sigma_{k|k-1}^2}. \tag{5.28}$$

And therefore,

$$\sigma_{k|k}^2 = \left[\frac{1}{\sigma_{k|k-1}^2} + p_{k|k}(1 - p_{k|k}) + \frac{\gamma_1^2}{\sigma_v^2} + \frac{\delta_1^2}{\sigma_w^2}\right]^{-1}. \tag{5.29}$$

When x_k gives rise to a binary observation n_k and two continuous observations r_k and s_k, the update equations in the state estimation step are

$$x_{k|k} = x_{k|k-1} + \frac{\sigma_{k|k-1}^2}{\sigma_v^2\sigma_w^2 + \sigma_{k|k-1}^2(\gamma_1^2\sigma_w^2 + \delta_1^2\sigma_v^2)}$$
$$\times \Big[\sigma_v^2\sigma_w^2(n_k - p_{k|k}) + \gamma_1\sigma_w^2(r_k - \gamma_0 - \gamma_1 x_{k|k-1})$$
$$+ \delta_1\sigma_v^2(s_k - \delta_0 - \delta_1 x_{k|k-1})\Big] \tag{5.30}$$

$$\sigma_{k|k}^2 = \left[\frac{1}{\sigma_{k|k-1}^2} + p_{k|k}(1 - p_{k|k}) + \frac{\gamma_1^2}{\sigma_v^2} + \frac{\delta_1^2}{\sigma_w^2}\right]^{-1}. \tag{5.31}$$

5.3 Deriving the Parameter Estimation Step Equations

In this state-space model, we have the parameters α, ρ, γ_0, γ_1, δ_0, δ_1, σ_v^2, and σ_w^2 to determine. We have already seen how γ_0, γ_1, and σ_v^2 were derived in the previous chapter when we had r_k. We will not repeat those derivations here again. Instead, we will only consider the derivations related to the new model parameters or changes to the way that earlier model parameters were derived. We will use this same approach of not re-deriving previous equations in the chapters that follow as well.

5.3.1 Deriving the Terms in the State Equation

We now have both ρ and α in the state equation. To determine them at the parameter estimation step, we will take the partial derivatives of the log-likelihood term containing ρ and α. In this case, the term we are interested in is

$$Q_1 = \frac{1}{2\sigma_\varepsilon^2} \sum_{k=1}^{K} \mathbb{E}\Big[(x_k - \rho x_{k-1} - \alpha I_k)^2\Big]. \tag{5.32}$$

Again, we set $x_0 = x_1$ to permit some bias at the beginning and ignore the relationship through ρ for this boundary condition. Therefore,

$$Q_1 = \frac{1}{2\sigma_\varepsilon^2} \left\{ \sum_{k=2}^{K} \mathbb{E}\Big[(x_k - \rho x_{k-1} - \alpha I_k)^2\Big] + \mathbb{E}\Big[(\alpha I_1)^2\Big] \right\}. \tag{5.33}$$

We will now take the partial derivatives of Q_1 with respect to α and ρ and set them to 0. Let us first begin with α.

$$\frac{\partial Q_1}{\partial \alpha} = \frac{1}{2\sigma_\varepsilon^2} \left\{ \sum_{k=2}^{K} \mathbb{E}\big[-2I_k(x_k - \rho x_{k-1} - \alpha I_k)\big] + 2\alpha I_1^2 \right\} = 0 \tag{5.34}$$

$$\begin{aligned} \implies 0 &= -\sum_{k=2}^{K} I_k \mathbb{E}\big[x_k\big] + \rho \sum_{k=2}^{K} I_k \mathbb{E}\big[x_{k-1}\big] + \alpha \sum_{k=1}^{K} I_k^2 \\ &= -\sum_{k=2}^{K} I_k x_{k|K} + \rho \sum_{k=2}^{K} I_k x_{k-1|K} + \alpha \sum_{k=1}^{K} I_k^2 \end{aligned} \tag{5.35}$$

$$\implies \rho \sum_{k=2}^{K} I_k x_{k-1|K} + \alpha \sum_{k=1}^{K} I_k^2 = \sum_{k=2}^{K} I_k x_{k|K}. \tag{5.36}$$

We will next take the partial derivative of Q_1 with respect to ρ.

$$\frac{\partial Q_1}{\partial \rho} = \frac{1}{2\sigma_\varepsilon^2} \sum_{k=2}^{K} \mathbb{E}\big[-2x_{k-1}(x_k - \rho x_{k-1} - \alpha I_k)\big] = 0 \tag{5.37}$$

$$\begin{aligned} \implies 0 &= -\sum_{k=2}^{K} \mathbb{E}\big[x_k x_{k-1}\big] + \rho \sum_{k=2}^{K} \mathbb{E}\big[x_{k-1}^2\big] + \alpha \sum_{k=2}^{K} I_k \mathbb{E}\big[x_{k-1}\big] \\ &= -\sum_{k=1}^{K-1} U_{k,k+1} + \rho \sum_{k=1}^{K-1} U_k + \alpha \sum_{k=2}^{K} I_k x_{k-1|K} \end{aligned} \tag{5.38}$$

$$\implies \rho \sum_{k=1}^{K-1} U_k + \alpha \sum_{k=2}^{K} I_k x_{k-1|K} = \sum_{k=1}^{K-1} U_{k,k+1}. \tag{5.39}$$

We now have two equations with which to solve for α and ρ.

The parameter estimation step updates for ρ and α when x_k evolves with time following $x_k = \rho x_{k-1} + \alpha I_k + \varepsilon_k$ are

$$\begin{bmatrix} \rho \\ \alpha \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{K-1} U_k & \sum_{k=2}^{K} I_k x_{k-1|K} \\ \sum_{k=2}^{K} I_k x_{k-1|K} & \sum_{k=1}^{K} I_k^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{k=1}^{K-1} U_{k,k+1} \\ \sum_{k=2}^{K} I_k x_{k|K} \end{bmatrix}. \tag{5.40}$$

5.3.2 *Deriving the Process Noise Variance*

We next consider the parameter estimation step update for the process noise variance σ_ε^2. The log-likelihood term containing σ_ε^2 is

$$\begin{aligned} Q_2 &= \frac{-K}{2} \log\left(2\pi\sigma_\varepsilon^2\right) - \sum_{k=1}^{K} \frac{\mathbb{E}\Big[(x_k - \rho x_{k-1} - \alpha I_k)^2\Big]}{2\sigma_\varepsilon^2} \\ &= \frac{-K}{2} \log\left(2\pi\sigma_\varepsilon^2\right) - \sum_{k=2}^{K} \frac{\mathbb{E}\Big[(x_k - \rho x_{k-1} - \alpha I_k)^2\Big]}{2\sigma_\varepsilon^2} - \frac{\mathbb{E}\Big[(\alpha I_1)^2\Big]}{2\sigma_\varepsilon^2}. \end{aligned} \tag{5.41}$$

We take the partial derivative of Q_2 with respect to σ_ε^2 and set it to 0 to solve for the parameter estimation step update.

$$\frac{\partial Q_2}{\partial \sigma_\varepsilon^2} = \frac{-K}{2\sigma_\varepsilon^2} + \frac{1}{2\sigma_\varepsilon^4}\sum_{k=2}^{K}\mathbb{E}\Big[(x_k - \rho x_{k-1} - \alpha I_k)^2\Big] + \frac{(\alpha I)^2}{2\sigma_\varepsilon^4} = 0 \tag{5.42}$$

$$\begin{aligned}\sigma_\varepsilon^2 &= \frac{1}{K}\sum_{k=2}^{K}\Big\{\mathbb{E}\big[x_k^2\big] - 2\rho\mathbb{E}\big[x_k x_{k-1}\big] + \rho^2\mathbb{E}\big[x_{k-1}^2\big] - 2\alpha I_k\mathbb{E}\big[x_k\big] \\ &\quad + 2\alpha\rho I_k\mathbb{E}\big[x_{k-1}\big]\Big\} + \frac{\alpha^2}{K}\sum_{k=1}^{K} I_k^2 \\ &= \frac{1}{K}\Big\{\sum_{k=2}^{K} U_k - 2\rho\sum_{k=1}^{K-1} U_{k,k+1} + \rho^2\sum_{k=1}^{K-1} U_k - 2\alpha\sum_{k=2}^{K} I_k x_{k|K} \\ &\quad + 2\alpha\rho\sum_{k=2}^{K} I_k x_{k-1|K} + \alpha^2\sum_{k=1}^{K} I_k^2\Big\}.\end{aligned} \tag{5.43}$$

The parameter estimation step update for σ_ε^2 when x_k evolves with time following $x_k = \rho x_{k-1} + \alpha I_k + \varepsilon_k$ is

$$\begin{aligned}\sigma_\varepsilon^2 &= \frac{1}{K}\Big\{\sum_{k=2}^{K} U_k - 2\rho\sum_{k=1}^{K-1} U_{k,k+1} + \rho^2\sum_{k=1}^{K-1} U_k - 2\alpha\sum_{k=2}^{K} I_k x_{k|K} \\ &\quad + 2\alpha\rho\sum_{k=2}^{K} I_k x_{k-1|K} + \alpha^2\sum_{k=1}^{K} I_k^2\Big\}.\end{aligned} \tag{5.44}$$

5.3.3 *Deriving the Constant Coefficient Terms and the Sensor Noise Variance*

The procedure for deriving the constant coefficients δ_0 and δ_1 related to s_k is similar to what we have seen earlier for γ_0 and γ_1. The derivation of the sensor noise variance σ_w^2 is also similar to that for σ_v^2.

The parameter estimation step updates for δ_0, δ_1, and σ_w^2 when we observe a second continuous variable $s_k = \delta_0 + \delta_1 x_k + w_k$ are

(continued)

$$\begin{bmatrix} \delta_0 \\ \delta_1 \end{bmatrix} = \begin{bmatrix} K & \sum_{k=1}^{K} x_{k|K} \\ \sum_{k=1}^{K} x_{k|K} & \sum_{k=1}^{K} U_k \end{bmatrix}^{-1} \begin{bmatrix} \sum_{k=1}^{K} s_k \\ \sum_{k=1}^{K} s_k x_{k|K} \end{bmatrix} \tag{5.45}$$

$$\sigma_w^2 = \frac{1}{K}\left\{ \sum_{k=1}^{K} s_k^2 + K\delta_0^2 + \delta_1^2 \sum_{k=1}^{K} U_k - 2\delta_0 \sum_{k=1}^{K} s_k - 2\delta_1 \sum_{k=1}^{K} s_k x_{k|K} + 2\delta_0\delta_1 \sum_{k=1}^{K} x_{k|K} \right\}. \tag{5.46}$$

5.4 MATLAB Examples

The code examples implementing the EM algorithm for the current state-space model are provided in the "one_bin_two_cont\sim" and "one_bin_two_cont\expm" folders. These two directories contain the following files:

- one_bin_two_cont\

  sim\

  data_one_bin_two_cont.mat
  filter_one_bin_two_cont.m
  data_one_bin_two_cont_no_extern_stim.mat
  filter_one_bin_two_cont_no_extern_stim.m

  expm\

  expm_data_one_bin_two_cont.mat
  expm_filter_one_bin_two_cont.m
  expm_data_one_bin_two_cont_no_extern_stim.mat
  expm_filter_one_bin_two_cont_no_extern_stim.m

For both simulated and experimental data, we have provided examples with and without αI_k (the external input). Results from running the code on a simulated example with αI_k included and on an experimental data example without αI_k are shown in Fig. 5.1. For simulated and experimental data containing αI_k, the ".m" files are named "filter_one_bin_two_cont.m" and "expm_filter_one_bin_two_cont.m," respectively. The corresponding examples *without* αI_k have the "no_extern_stim" suffix added to them.

In this case, the model takes in as inputs the variables `n`, `r`, and `s` that denote n_k, r_k, and s_k, respectively, for estimating x_k. Since there are three different observations, the code also has more parameters to initialize. In the code, the

variables r0 and r1 are used for γ_0 and γ_1, respectively, and s0 and s1 are used for δ_0 and δ_1. The variables vr and vs denote the corresponding sensor noise variances σ_v^2 and σ_w^2. Finally, ve, rho, and alpha denote the process noise variance σ_ε^2, the forgetting factor ρ, and the α term related to I_k, respectively. Shown below is a brief code snippet showing the parameter initialization.

```
r0(1) = r(1);
r1(1)  = 0.5;
s0(1) = s(1);
s1(1) = 1;
vr(1) = 0.05;
vs(1) = 0.05;
ve(1) = 0.05;
rho(1) = 1;
alpha(1) = 0.5;
```

Also, base_prob (p_0) is still calculated based on the average probability of $n_k = 1$ occurring in the data. The other variables x_pred, x_updt, and x_smth for $x_{k|k-1}$, $x_{k|k}$, and $x_{k|K}$ remain the same, as well as the corresponding v_pred, v_updt and v_smth variables for variance. There is a sequential progression in the code through $k = 1, 2, \ldots, K$ and then through $k = K, (K-1), \ldots, 1$ at the state estimation step. The terms r0, r1, s0, s1, vr, vs, ve, rho, and alpha are calculated at the parameter estimation step. Shown below is a code snippet in the forward progression.

```
x_pred(k) = rho(m) * x_updt(k - 1) + alpha(m) * I(k);
v_pred(k) = (rho(m) ^ 2) * v_updt(k - 1) + ve(m);

C(k) = v_pred(k) / (vr(m) * vs(m) + v_pred(k) * ((r1(m) ^ 2) * vs
    (m) + (s1(m) ^ 2) * vr(m)));
x_updt(k) = get_posterior_mode(x_pred(k), C(k), r(k), r0(m), r1(m
    ), b0(m), b1(m), vr(m), n(k), s(k), s0(m), s1(m), vs(m));
p_updt(k) = 1 / (1 + exp((-1) * (b0(m) + b1(m) * x_updt(k))));
v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(m)) + ((s1(m
    ) ^ 2) / vs(m)) + (b1(m) ^ 2) * p_updt(k) * (1 - p_updt(k)));
```

The code where we proceed in the reverse direction at the state estimation step is shown below. While it is largely similar to what we saw in an earlier chapter, now the variable rho is also included.

```
x_smth(K) = x_updt(K);
v_smth(K) = v_updt(K);
W(K) = v_smth(K) + (x_smth(K) ^ 2);

A(1:(end - 1)) = rho(m) * v_updt(1:(end - 1)) ./ v_pred(2:end);

for k = (K - 1):(-1):1
  x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k + 1));
  v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) - v_pred(k
    + 1));

  CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
  W(k) = v_smth(k) + (x_smth(k) ^ 2);
end
```

Note that in the examples where an external input is absent, `alpha` is not calculated. The state estimation step and the parameter estimation step are performed in turn until convergence.

5.4.1 Application to Skin Conductance and Sympathetic Arousal

This state-space model with one binary and two continuous observations was used in [29] for estimating sympathetic arousal from skin conductance. In the model, the tonic component made up the continuous variable s_k. The other continuous variable r_k was derived somewhat differently. SCR amplitudes can have a skewed distribution which a log transformation can help correct. Therefore, the log of the SCR amplitudes was taken and interpolated over to generate r_k.

A further point is to be noted with experimental data. The estimated state x_k can occasionally overfit to one of the continuous variables [29]. Consequently, an additional constraint was applied to allow the parameters corresponding to r_k and s_k (i.e., γ_0, γ_1, σ_v^2, δ_0, δ_1, and σ_w^2) to update only if the sensor noise variance estimates did not differ by more than a certain amount. Details of this can be found in [29]. This constraint prevented one of the sensor noise variance estimates from being driven down at the expense of the other (which takes place during overfitting).

If the external inputs are unknown, the version of the code without αI_k can be used. The experimental results in Fig. 5.1 are from a case where αI_k is not considered. The data come from the stress experiment in [53] which we also considered two chapters ago. The portion of the experiment considered here consists of the cognitive stressors, relaxation, and the horror movie clip. The state estimates are high during the cognitive stressors and thereafter gradually diminish. However, the increase seen in the HAI at the beginning of the horror movie clip is quite significant.

Data from the Pavlovian fear conditioning experiment in [66] are taken for the experimental code example containing the αI_k term. The results are shown in Fig. 5.2. The experiment is described in detail in [67, 68]. In a typical fear conditioning experiment, a neutral cue is paired with an unpleasant stimulus such as a painful electric shock. Through repeated pairing, a subject begins to display

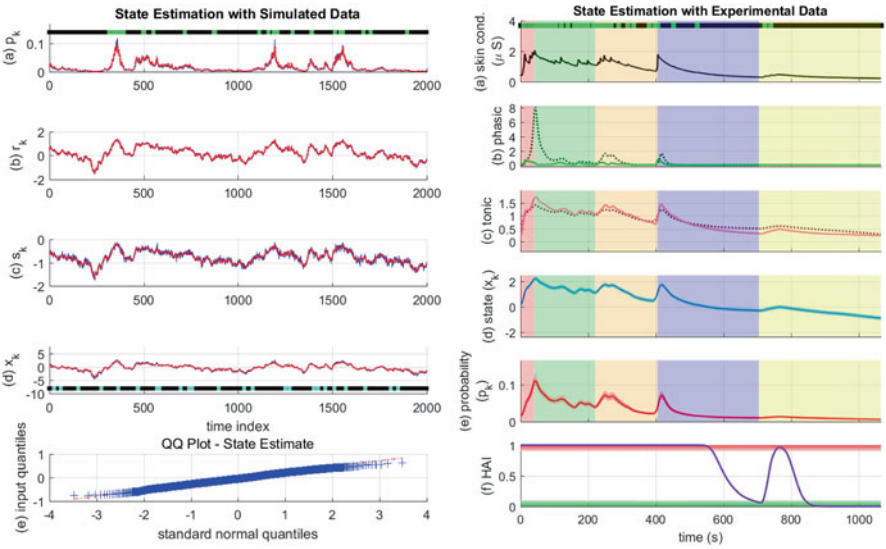

Fig. 5.1 State estimation based on observing one binary and two continuous variables. The left sub-figure depicts estimation on simulated data, and the right sub-figure depicts the estimation of sympathetic arousal from skin conductance data. The sub-panels on the left, respectively, depict: (**a**) the probability of binary event occurrence p_k (blue) and its estimate (red) (the green and black dots above at the top denote the presence or absence of binary events, respectively); (**b**) the first continuous variable r_k (blue) and its estimate (red); (**c**) the second continuous variable s_k (blue) and its estimate (red); (**d**) the state x_k (blue) and its estimate (red) (the cyan and black dots denote the presence or absence of external binary inputs, respectively); (**e**) the QQ plot for the residual error of x_k. The sub-panels on the right, respectively, depict: (**a**) the skin conductance signal (the green and black dots on top denote the presence or absence of SCRs, respectively); (**b**) the phasic-derived variable r_k (green solid) and its estimate (dotted); (**c**) the tonic level s_k (pink solid) and its estimate (dotted); (**d**) the arousal state x_k and its 95% confidence limits; (**e**) the probability of SCR occurrence p_k and its 95% confidence limits; (**f**) the HAI (the regions above 90% and below 10% are shaded in red and green, respectively). The background colors on the right sub-figure correspond to the instruction period, a counting task, a color–word association task, relaxation, and watching a horror movie clip. From [32], used under Creative Commons CC-BY license

a response to the neutral cue alone. In the experiment in [66], two types of cues were used. One of the cues never preceded the electric shock. This is labeled the CS- cue. The second cue, labeled as CS+, preceded the shock 50% of the time. The code example sets $I_k = 1$ at the locations of the neutral cues and the shocks. Other types of inputs may also be considered for I_k. Figure 5.2 depicts the averages for the CS- trials, the CS+ trials that did not contain the electric shock, and the CS+ trials that did contain the shock. As seen in Fig. 5.2, for this particular subject, the CS+ with the shock elicited the highest skin conductance and sympathetic arousal responses. The CS- trials had the lowest skin conductance and arousal levels, and the CS+ without the shock had responses in-between these two.

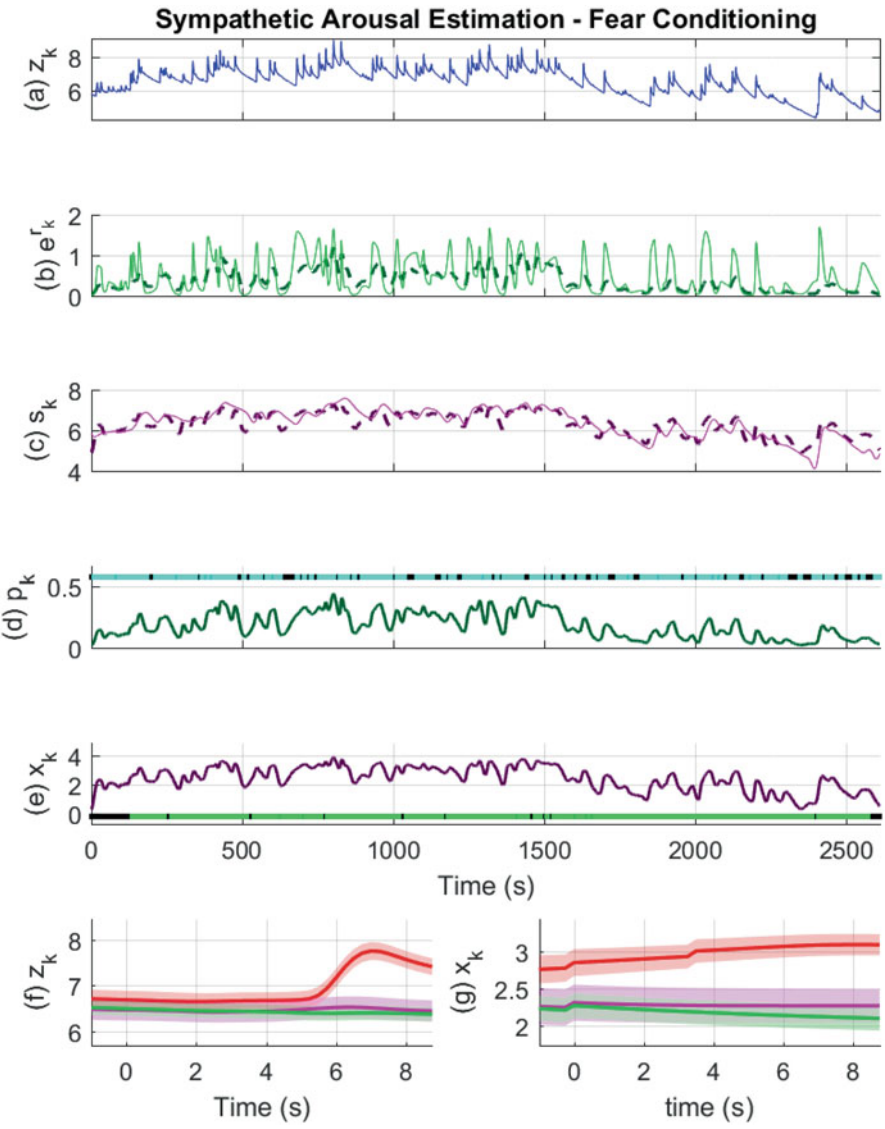

Fig. 5.2 State estimation in Pavlovian fear conditioning. The sub-panels, respectively, depict: (**a**) the skin conductance signal z_k; (**b**) the phasic-derived variable (green solid) and its estimate (dashed); (**c**) the tonic level s_k (mauve solid) and its estimate (dashed); (**d**) the probability of SCR occurrence p_k (the cyan and black dots on top denote the presence or absence of SCRs, respectively); (**e**) the arousal state x_k (the green and black dots denote the presence or absence of external binary inputs, respectively); (**f**) the averages corresponding to different trials for skin conductance (CS− —green, CS+ without the shock—mauve and CS+ with the shock—red); (**g**) the same averages for the arousal state x_k. © 2020 IEEE. Reprinted, with permission, from [29]

Chapter 6
State-Space Model with One Binary, Two Continuous, and a Spiking-Type Observation

Spiking-type observations are occasionally recorded in experiments. For instance, neural spiking activity may be recorded from a macaque monkey engaged in a learning experiment or an EKG signal may be recorded from a human subject in an experiment. In such instances, we can model the spiking-type variable using a *conditional intensity function* (CIF). The CIF is similar to the rate parameter in a Poisson distribution but is more general. With spiking-type observations, we usually assume that our state variable x_k affects the *rate* of spiking through the CIF. Now we need to estimate x_k at each time index k. In the case of a spiking-type variable, we typically observe the spiking over a *short interval* corresponding to time index k. For instance, in the case of a macaque monkey performing a behavioral learning task, we may observe neural spiking over a period of several hundred milliseconds corresponding to each trial k. Each trial duration is then divided into smaller bins indexed over j. Since the spiking-type variables are binary, we assign either $m_{k,j} = 0$ or $m_{k,j} = 1$ within the interval k for each of the smaller time bins j based on spike occurrence. Shown below is an example CIF $\lambda_{k,j}$ used in an experiment where a monkey's learning state was estimated from measurements that included neural spiking [6].

$$\lambda_{k,j} = e^{\theta_0 + \psi x_k + \sum_{s=1}^{S} \theta_s m_{k,j-s}}. \tag{6.1}$$

In general, the specific form of the CIF depends on the type of application. In this chapter, we will derive the state and parameter estimation step equations for a model where a spiking-type variable characterized by a general CIF $\lambda_{k,j}$ is observed along

Supplementary Information The online version contains supplementary material available at (https://doi.org/10.1007/978-3-031-47104-9_6).

GitHub https://github.com/computational-medicine-lab/A-Tutorial-on-the-Design-of-Bayesian-Filters/tree/main/one_bin_two_cont_one_spk

D. S. Wickramasuriya, R. T. Faghih, *Bayesian Filter Design for Computational Medicine*, https://doi.org/10.1007/978-3-031-47104-9_6

with one binary and two continuous variables. We will, however, first consider the need for such a state-space model.

In the preceding chapter, we looked at a state-space model for estimating sympathetic arousal based on one binary and two continuous skin conductance observations. The occurrence of SCRs made up the binary observation n_k. The continuous observations comprised of a transformed version of the SCR peaks and the tonic level. In reality, the sympathetic nervous system affects a number of organs, not just the skin. We can go to any one of these organs to extract features to estimate arousal. However, not all these organs (or the corresponding physiological signals) are conveniently accessible. The heart is one organ affected by sympathetic activation for which the corresponding signals can be measured easily (e.g., using an EKG). Now sympathetic drive is known to increase heart rate and the force of ventricular contraction [69]. The heart, however, is innervated by both sympathetic and parasympathetic fibers and also has its own pacing mechanism [70]. Consequently, a precise extraction of the sympathetic activation component from an EKG signal is a challenge. In [31], a state-space model based on three skin conductance features (the features just referred to) and EKG signals modeled as spiking observations was used to estimate sympathetic arousal. Here, the model assumed that increased sympathetic arousal caused EKG inter-beat intervals (known as RR-intervals) to decrease (i.e., caused heart rate to increase). The CIF was based on the history-dependent inverse Gaussian (HDIG) probability density function for RR-intervals [71, 72]. The state-space model could be used for wearable healthcare applications (Fig. 6.1). Post-traumatic stress disorder (PTSD), for instance, is known to involve symptoms of *hyper*arousal [73], while major depression is known to involve low levels of arousal [74]. Thus, a wearable device based on skin

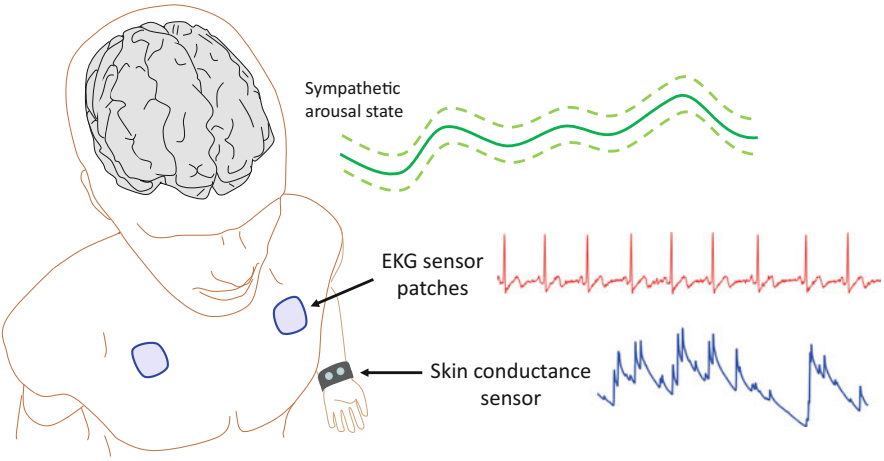

Fig. 6.1 A wearable sensing system for decoding sympathetic arousal. The sweat glands are innervated by sympathetic nerve fibers, and the heart is innervated by both sympathetic and parasympathetic fibers. This information from skin conductance and heart rate can be used to estimate sympathetic arousal. From [26], used under Creative Commons CC-BY license

conductance and heart rate measurements for monitoring arousal could be used to help care for such patients.

We also make another notable observation here. The phenomena occurring within the human body and brain are rather complex. Thus, it is likely that no *single* type of physiological signal or feature captures all the necessary information regarding latent physiological states. If, for instance, both emotional valence and arousal are to be decoded, features from a number of signals could be considered [58, 75–78]. Signals such as EMG [27, 79–82], heart rate [83–87], respiration [88–92], and blood flow signals within the brain (functional near infrared spectroscopy) [93–97] all contain information regarding phenomena such as emotion and cognitive effort.

6.1 Deriving the Predict Equations in the State Estimation Step

We have already considered three different cases for the state equation: (i) the simple random walk; (ii) the random walk with a forgetting factor ρ; (iii) the random walk with a forgetting factor ρ and an external input I_k. You would have noticed by now that changes to the state equation primarily affect the predict equations in the state estimation step and not the update equations. The three cases we have considered thus far cover most of the applications that are encountered in typical physiological state estimation problems. In the current state-space model, we will assume that x_k evolves with time following one of the state equations we have already seen. Thus no new predict step equations have to be derived. These signals could be used for wearable healthcare applications. A study of how different external stimuli also affect emotion could lead to novel neuromarketing strategies as well [98].

6.2 Deriving the Update Equations in the State Estimation Step

When dealing with a spiking-type observation, we first split our observation interval at time index k into smaller segments and index these smaller bins as $j = 1, 2, \ldots, J$. The joint probability of the spikes over the J observation bins is then [99]

$$p(m_{k,1}, m_{k,2}, \ldots, m_{k,J}|x_k) = e^{\sum_{j=1}^{J} \log(\lambda_{k,j}\Delta)m_{k,j} - \lambda_{k,j}\Delta}. \tag{6.2}$$

Recall from (5.18) that when we had one binary and two continuous observations, the posterior density was

$$p(x_k|y_{1:k}) \propto p(n_k|x_k)p(r_k|x_k)p(s_k|x_k)p(x_k|n_{1:k-1}, r_{1:k-1}, s_{1:k-1}). \tag{6.3}$$

Now that we have the spiking-type observation, we will include $p(m_{k,1}, m_{k,2}, \ldots, m_{k,J}|x_k)$ in $p(x_k|y_{1:k})$ as well. Therefore,

$$p(x_k|y_{1:k}) \propto e^{n_k \log(p_k)+(1-n_k)\log(1-p_k)} \times e^{\frac{-(r_k-\gamma_0-\gamma_1 x_k)^2}{2\sigma_v^2}} \times e^{\frac{-(s_k-\delta_0-\delta_1 x_k)^2}{2\sigma_w^2}} \times e^{\sum_{j=1}^{J} \log(\lambda_{k,j}\Delta)m_{k,j}-\lambda_{k,j}\Delta} \times e^{\frac{-(x_k-x_{k|k-1})^2}{2\sigma_{k|k-1}^2}}. \tag{6.4}$$

The procedure for deriving the update equations is again similar to what we have seen thus far. As before, we will take the first derivative of the exponent term, set it to 0, and solve for x_k to obtain the mean. We will then take the second derivative to obtain the uncertainty or variance associated with the estimate. Taking the log of the posterior density and setting the first partial derivative to 0 yield

$$\begin{aligned}\frac{dq}{dx_k} =& \frac{-(x_k - x_{k|k-1})}{\sigma_{k|k-1}^2} + (n_k - p_k) + \frac{\gamma_1(r_k - \gamma_0 - \gamma_1 x_k)}{\sigma_v^2} \\ &+ \frac{\delta_1(s_k - \delta_0 - \delta_1 x_k)}{\sigma_w^2} + \sum_{j=1}^{J} \frac{1}{\lambda_{k,j}} \frac{d\lambda_{k,j}}{dx_k}(m_{k,j} - \lambda_{k,j}\Delta) = 0.\end{aligned} \tag{6.5}$$

Solving for x_k is now similar to what we saw in the earlier chapter. We simply need to add and subtract $\gamma_1 x_{k|k-1}$ and $\delta_1 x_{k|k-1}$ from the terms containing r_k and s_k, respectively. The second partial derivative is

$$\begin{aligned}\frac{d^2q}{dx_k^2} &= \frac{-1}{\sigma_{k|k-1}^2} - \frac{dp_k}{dx_k} - \frac{\gamma_1^2}{\sigma_v^2} - \frac{\delta_1^2}{\sigma_w^2} + \frac{d}{dx_k}\left[\sum_{j=1}^{J} \frac{1}{\lambda_{k,j}} \frac{d\lambda_{k,j}}{dx_k}(m_{k,j} - \lambda_{k,j}\Delta)\right] \\ &= \frac{-1}{\sigma_{k|k-1}^2} - p_k(1-p_k) - \frac{\gamma_1^2}{\sigma_v^2} - \frac{\delta_1^2}{\sigma_w^2} \\ &\quad + \sum_{j=1}^{J}\left[\frac{1}{\lambda_{k,j}} \frac{d^2\lambda_{k,j}}{dx_k^2}(m_{k,j} - \lambda_{k,j}\Delta) - \frac{m_{k,j}}{\lambda_{k,j|k}^2}\left(\frac{d\lambda_{k,j}}{dx_k}\right)^2\right].\end{aligned} \tag{6.6}$$

Thus the updates for $x_{k|k}$ and $\sigma_{k|k}^2$ turn out to be

$$\begin{aligned}x_{k|k} = x_{k|k-1} &+ \frac{\sigma_{k|k-1}^2}{\sigma_v^2\sigma_w^2 + \sigma_{k|k-1}^2(\gamma_1^2\sigma_w^2 + \delta_1^2\sigma_v^2)}\Bigg[\sigma_v^2\sigma_w^2(n_k - p_{k|k}) \\ &+ \gamma_1\sigma_w^2(r_k - \gamma_0 - \gamma_1 x_{k|k-1}) + \delta_1\sigma_v^2(s_k - \delta_0 - \delta_1 x_{k|k-1}) \\ &+ \sigma_v^2\sigma_w^2 \sum_{j=1}^{J} \frac{1}{\lambda_{k,j|k}} \frac{d\lambda_{k,j|k}}{dx_k}(m_{k,j} - \lambda_{k,j|k}\Delta)\Bigg]\end{aligned} \tag{6.7}$$

$$\sigma_{k|k}^2 = \left\{ \frac{1}{\sigma_{k|k-1}^2} + p_{k|k}(1-p_{k|k}) + \frac{\gamma_1^2}{\sigma_v^2} + \frac{\delta_1^2}{\sigma_w^2} - \sum_{j=1}^{J} \left[\frac{1}{\lambda_{k,j|k}} \frac{d^2\lambda_{k,j|k}}{dx_k^2}(m_{k,j} - \lambda_{k,j|k}\Delta) - \frac{m_{k,j}}{\lambda_{k,j|k}^2}\left(\frac{d\lambda_{k,j|k}}{dx_k}\right)^2 \right] \right\}^{-1}. \tag{6.8}$$

Note that the equations may simplify further depending on the specific form of the CIF. Here we have provided the derivations for the general case.

When x_k gives rise to a binary observation n_k, two continuous observations r_k and s_k and a spiking-type observation $m_{k,j}$ characterized by the CIF $\lambda_{k,j}$, the update equations in the state estimation step are

$$\begin{aligned} x_{k|k} = x_{k|k-1} + \frac{\sigma_{k|k-1}^2}{\sigma_v^2\sigma_w^2 + \sigma_{k|k-1}^2(\gamma_1^2\sigma_w^2 + \delta_1^2\sigma_v^2)} \Bigg[& \sigma_v^2\sigma_w^2(n_k - p_{k|k}) \\ & + \gamma_1\sigma_w^2(r_k - \gamma_0 - \gamma_1 x_{k|k-1}) + \delta_1\sigma_v^2(s_k - \delta_0 - \delta_1 x_{k|k-1}) \\ & + \sigma_v^2\sigma_w^2 \sum_{j=1}^{J} \frac{1}{\lambda_{k,j|k}} \frac{d\lambda_{k,j|k}}{dx_k}(m_{k,j} - \lambda_{k,j|k}\Delta) \Bigg] \end{aligned} \tag{6.9}$$

$$\begin{aligned} \sigma_{k|k}^2 = \Bigg\{ \frac{1}{\sigma_{k|k-1}^2} + p_{k|k}(1-p_{k|k}) + \frac{\gamma_1^2}{\sigma_v^2} + \frac{\delta_1^2}{\sigma_w^2} - \sum_{j=1}^{J} \Bigg[& \frac{1}{\lambda_{k,j|k}} \frac{d^2\lambda_{k,j|k}}{dx_k^2} \\ & (m_{k,j} - \lambda_{k,j|k}\Delta) - \frac{m_{k,j}}{\lambda_{k,j|k}^2}\left(\frac{d\lambda_{k,j|k}}{dx_k}\right)^2 \Bigg] \Bigg\}^{-1}. \end{aligned} \tag{6.10}$$

6.3 Deriving the Parameter Estimation Step Equations

The state-space model we consider here is an extension of what we considered in the previous chapter that contained one binary and two continuous observations. Therefore, the only new parameter estimation step equations we need to derive are for the spiking-type variable.

6.3.1 Deriving the Coefficients Within a CIF

A CIF can take different forms depending on the type of application. For instance, when neural spiking data are involved, $\log(\lambda_{k,j})$ may be expressed as a linear sum of history-dependent terms and x_k as in (6.1). If this is the case, we would have to determine ψ and the θ_s's at the parameter estimation step. When heartbeats are modeled as a spiking-type variable, the CIF involves an inverse Gaussian distribution and could be related to x_k through its mean [31]. Thus, the terms to be derived at the parameter estimation step when a spiking-type variable is present are application-specific. In general, due to the rather complicated nature of a CIF, the parameter estimation step updates do not have neat closed-form expressions. Instead, the parameters have to be chosen to maximize the expected log-likelihood

$$Q = \sum_{k=1}^{K} \sum_{j=1}^{J} \mathbb{E}\Big[\log(\lambda_{k,j}\Delta) m_{k,j} - \lambda_{k,j}\Delta \Big]. \tag{6.11}$$

The form of Q can be deduced from (6.2). The trick to maximizing Q is to perform a Taylor expansion around the mean $x_{k|K} = \mathbb{E}[x_k]$ for each of the summed terms. Therefore, when the expected value is finally calculated, we will end up with terms like $\mathbb{E}[x_k - x_{k|K}]$ and $\mathbb{E}[(x_k - x_{k|K})^2]$ in the expansion. Now

$$\mathbb{E}[x_k - x_{k|K}] = \mathbb{E}[x_k] - x_{k|K} \quad \text{based on (2.2)} \tag{6.12}$$

$$= x_{k|K} - x_{k|K} = 0, \tag{6.13}$$

and $\mathbb{E}[(x_k - x_{k|K})^2]$ is the variance $\sigma^2_{k|K}$. These two facts will greatly help simplify the calculation of Q.

Let us now perform the Taylor expansion around $x_{k|K}$ [6]. The summed term within the expected value simplifies to

$$\begin{aligned} \log(\lambda_{k,j}\Delta) m_{k,j} - \lambda_{k,j}\Delta \approx & \log(\lambda_{k,j|K}\Delta) m_{k,j} - \lambda_{k,j|K}\Delta \\ & + \frac{1}{\lambda_{k,j|K}} \frac{\partial \lambda_{k,j|K}}{\partial x_k} (m_{k,j} - \lambda_{k,j|K}\Delta)(x_k - x_{k|K}) \\ & + \frac{1}{2}\Bigg[\frac{1}{\lambda_{k,j|K}} \frac{\partial^2 \lambda_{k,j|K}}{\partial x_k^2} (m_{k,j} - \lambda_{k,j|K}\Delta) \\ & - \frac{m_{k,j}}{\lambda^2_{k,j|K}} \left(\frac{\partial \lambda_{k,j|K}}{\partial x_k} \right)^2 \Bigg] (x_k - x_{k|K})^2. \end{aligned} \tag{6.14}$$

Taking the expected value, we have

$$\mathbb{E}\Big[\log(\lambda_{k,j}\Delta) m_{k,j} - \lambda_{k,j|K}\Delta \Big] \approx \log(\lambda_{k,j|K}\Delta) m_{k,j} - \lambda_{k,j|K}\Delta$$

$$+\frac{1}{\lambda_{k,j|K}}\frac{\partial\lambda_{k,j|K}}{\partial x_k}(m_{k,j}-\lambda_{k,j|K}\Delta)\mathbb{E}\big[x_k-x_{k|K}\big]$$
$$+\frac{1}{2}\Bigg[\frac{1}{\lambda_{k,j|K}}\frac{\partial^2\lambda_{k,j|K}}{\partial x_k^2}(m_{k,j}-\lambda_{k,j|K}\Delta)$$
$$-\frac{m_{k,j}}{\lambda_{k,j|K}^2}\left(\frac{\partial\lambda_{k,j|K}}{\partial x_k}\right)^2\Bigg]$$
$$\times\,\mathbb{E}\big[x_k-x_{k|K}\big]^2. \tag{6.15}$$

Note the terms $\mathbb{E}[x_k - x_{k|K}]$ and $\mathbb{E}[(x_k - x_{k|K})^2]$ in the expression above. The first of these is 0, and the second is the variance $\sigma^2_{k|K}$. Therefore,

$$\mathbb{E}\big[\log(\lambda_{k,j}\Delta)m_{k,j}-\lambda_{k,j|K}\Delta\big]\approx\log(\lambda_{k,j|K}\Delta)m_{k,j}-\lambda_{k,j|K}\Delta+0$$
$$+\frac{1}{2}\Bigg[\frac{1}{\lambda_{k,j|K}}\frac{\partial^2\lambda_{k,j|K}}{\partial x_k^2}(m_{k,j}-\lambda_{k,j|K}\Delta)$$
$$-\frac{m_{k,j}}{\lambda_{k,j|K}^2}\left(\frac{\partial\lambda_{k,j|K}}{\partial x_k}\right)^2\Bigg]\sigma^2_{k|K}. \tag{6.16}$$

Consequently, Q approximately simplifies to

$$Q\approx\sum_{k=1}^{K}\sum_{j=1}^{J}\log(\lambda_{k,j|K}\Delta)m_{k,j}-\lambda_{k,j|K}\Delta$$
$$+\frac{1}{2}\Bigg[\frac{1}{\lambda_{k,j|K}}\frac{\partial^2\lambda_{k,j|K}}{\partial x_k^2}(m_{k,j}-\lambda_{k,j|K}\Delta)-\frac{m_{k,j}}{\lambda_{k,j|K}^2}\left(\frac{\partial\lambda_{k,j|K}}{\partial x_k}\right)^2\Bigg]\sigma^2_{k|K}. \tag{6.17}$$

In general, Q will have to be maximized with respect to the model parameters in the CIF using numerical methods.

The parameter estimation step updates for the terms in a CIF $\lambda_{k,j}$ when we observe a spiking-type variable $m_{k,j}$ are chosen to maximize

$$Q\approx\sum_{k=1}^{K}\sum_{j=1}^{J}\log(\lambda_{k,j|K}\Delta)m_{k,j}-\lambda_{k,j|K}\Delta$$

(continued)

$$+\frac{1}{2}\left[\frac{1}{\lambda_{k,j|K}}\frac{\partial^2\lambda_{k,j|K}}{\partial x_k^2}(m_{k,j}-\lambda_{k,j|K}\Delta)-\frac{m_{k,j}}{\lambda_{k,j|K}^2}\left(\frac{\partial\lambda_{k,j|K}}{\partial x_k}\right)^2\right]\sigma_{k|K}^2. \tag{6.18}$$

6.4 MATLAB Examples

MATLAB code examples for simulated and experimental data for the state-space model with one binary, two continuous, and one spiking-type observation are provided in the folders shown below:

- one_bin_two_cont_one_spk

  sim\

  data_one_bin_two_cont_one_spk.mat
  filter_one_bin_two_cont_one_spk.m

  expm\

  expm_data_one_bin_two_cont_one_spk.mat
  expm_filter_one_bin_two_cont_one_spk.m

6.4.1 Application to Skin Conductance, Heart Rate and Sympathetic Arousal

The state-space model described in this chapter was used in [31] to estimate sympathetic arousal from skin conductance and heart rate measurements. The skin conductance observations are the same three that were used in [29] (discussed in the previous chapter). Thus, the only new observation added here relates to heart rate for which some additional discussion is necessary.

The code examples estimate arousal from the four observations related to skin conductance and heart rate. The R-peaks in the EKG signals are taken to form the spiking observations. If L consecutive R-peaks occur at times u_l within $(0, T]$ such that $0 < u_1 < u_2 < \ldots < u_L \leq T$, and $h_l = u_l - u_{l-1}$ is the l^{th} RR-interval, the HDIG density function for the RR-intervals at $t > u_l$ is

$$g(t|u_l) = \sqrt{\frac{\theta_{q+1}}{2\pi(t-u_l)^3}}\exp\left\{\frac{-\theta_{q+1}[t-u_l-\mu]^2}{2\mu^2(t-u_l)}\right\}, \tag{6.19}$$

where q is the model order, θ_{q+1} is related to the variance, and the mean is

$$\mu = \theta_0 + \sum_{i=1}^{q} \theta_i h_{l-i+1} + \eta x_k, \tag{6.20}$$

where η is a coefficient to be determined. Accordingly, a change in sympathetic arousal x_k causes the mean of the HDIG density function to shift (i.e., heart rate speeds up or slows down depending on the arousal level). The CIF $\lambda_{k,j}$ is

$$\lambda_{k,j} \triangleq \frac{g(t_{k,j}|u_{k,j})}{1 - \int_{u_{k,j}}^{t_{k,j}} g(z|u_{k,j})dz}, \tag{6.21}$$

where $u_{k,j}$ is the time of occurrence of the last R-peak prior to $t_{k,j}$. The CIF $\lambda_{k,j}$ is calculated every $\Delta = 5$ ms [23, 71]. Since skin conductance is typically analyzed at 4 Hz ($t_s = 250$ ms), there are $250/5 = 50$ smaller observation bins j for heart rate at each time index k. Due to computational complexity, the θ_i's were estimated separately in an offline manner using maximum likelihood. Now the work by Barbieri et al. [71] was one of the earliest to perform point process analysis of EKG RR-intervals using the HDIG density function.[1] The EM algorithm in [31] was executed for several different values of η, and the best one was selected based on a maximization of the log-likelihood term in (6.17). Note also that since the experimental code example involves skin conductance and heart rate with $\Delta = 5$ ms bins, the heart rate observations need to be provided to the code in a manner similar to that contained in the .mat file.

The other aspects of the code and the variable names are similar to what was described in earlier chapters. Running the code examples on simulated and experimental data yields the results shown in Fig. 6.2. The experimental data results are from the Pavlovian fear conditioning experiment in [100]. As shown in the figure, the CS+ trials with the electric shock have the highest average responses, while the CS- trials have the lowest average responses for the subject considered. The CS+ trials without the shock have an intermediate response.

[1] The code for calculating the θ_i's for a series of RR-interval measurements via maximum likelihood is provided at http://users.neurostat.mit.edu/barbieri/pphrv.

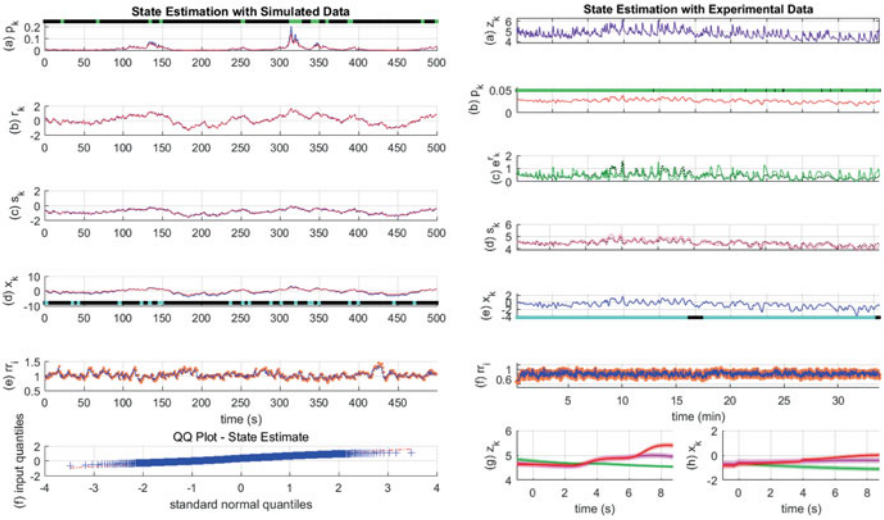

Fig. 6.2 State estimation based on observing one binary, two continuous, and one spiking-type variable. The left sub-figure depicts estimation on simulated data, and the right sub-figure depicts the estimation of sympathetic arousal from skin conductance and heart rate data. The sub-panels on the left, respectively, depict: (**a**) the probability of binary event occurrence p_k (blue) and its estimate (red) (the green and black dots above at the top denote the presence or absence of binary events, respectively); (**b**) the first continuous variable r_k (blue) and its estimate (red); (**c**) the second continuous variable s_k (blue) and its estimate (red); (**d**) the state x_k (blue) and its estimate (red) (the cyan and black dots denote the presence or absence of external binary inputs, respectively); (**e**) the simulated RR-interval sequence (orange) and the fit to the HDIG mean; (**f**) the QQ plot for the residual error of x_k. The sub-panels on the right, respectively, depict: (**a**) the skin conductance signal z_k; (**b**) the probability of SCR occurrence p_k (the green and black dots on top denote the presence or absence of SCRs, respectively); (**c**) the phasic-derived variable (green solid) and its estimate (dotted); (**d**) the tonic level s_k (pink solid) and its estimate (dotted); (**e**) the arousal state x_k (the cyan and black dots denote the presence or absence of external binary inputs, respectively); (**f**) the RR-interval sequence (orange) and the fit to the HDIG mean; (**g**) the averages corresponding to different trials for skin conductance (CS− —green, CS+ without the shock—mauve and CS+ with the shock—red); (**h**) the same averages for the state. From [31], used under Creative Commons CC-BY license

Chapter 7
State-Space Model with One Marked Point Process (MPP) Observation

Thus far we have considered binary observations and continuous observations in our state-space models. With binary observations, we do not consider the magnitudes of the binary-valued events (since each is just a 0 or a 1) but are merely interested in the event occurrences. Consequently, we can treat the spiking-type observations in the earlier chapter as binary-valued as well. There too, our concern was primarily with the occurrence of the cardiac contractions and the accompanying spikes in an EKG signal, but not the actual amplitudes of the spikes. But what happens when we observe a point process that is not just a sequence of zeros and ones but rather is a sequence of zeros and *real-valued amplitudes*? Such a point process forms a marked point process (MPP). These are encountered frequently in physiological state estimation applications as well. For instance, the sequence of neural impulses underlying a skin conductance signal forms an MPP (Fig. 3.2). So do pulsatile hormone secretory events. In this chapter, we will learn how to derive the state and parameter estimation step equations when the state-space model contains MPP observations.

In this chapter also, we will begin by considering a motivating example. Now we can build many models ranging from simple to complex to account for physiological phenomena. Any mathematical abstraction of a real-world system will have some imperfections to it and will not be able to fully account for all of the data. Occasionally, in engineering systems, we will encounter cases where a simpler model performs better than a more complex model. The estimation of sympathetic arousal from skin conductance is one such case. The state-space model with one binary and two continuous observations is quite complex [29]. However, despite its complexity, it is somewhat imperfect in that it interpolates over a log-transformed

Supplementary Information The online version contains supplementary material available at (https://doi.org/10.1007/978-3-031-47104-9_7).

GitHub https://github.com/computational-medicine-lab/A-Tutorial-on-the-Design-of-Bayesian-Filters/tree/main/one_mpp

D. S. Wickramasuriya, R. T. Faghih, *Bayesian Filter Design for Computational Medicine*, https://doi.org/10.1007/978-3-031-47104-9_7

version of the SCR amplitudes. A more natural way to account for phasic skin conductance variations is to model the underlying neural impulses as an MPP [32]. This eliminates the need for *two* continuous variables and is simpler.

A further algorithmic detail is also worth noting here. Mathematical models of real-world systems will always have some limitations. The limitations may be in the model itself or have to do with issues that arise during computation. This book focuses on the estimation of unobserved physiological quantities that are related (fully or partially) to point process observations. Occasionally, when we have both binary and continuous variables involved, the EM algorithm can have a tendency to converge to locations where there is a near-perfect fit to one of the continuous variables (i.e., overfitting occurs). The state-space model with one binary and two continuous observations has this tendency to overfit on experimental data. Consequently, additional constraints have to be put in place to control it [29]. This issue can also occur in the model with one binary and *one* continuous observation. The use of the MPP framework circumvents the need to have a continuous variable and thus avoids the need for external overfitting control. Thus the simpler MPP state-space model for estimating arousal based on skin conductance performed quite well in comparison to others [32].

7.1 Deriving the Update Equations in the State Estimation Step

In this chapter also, we will assume that x_k evolves with time following one of the state equations we have already seen. Thus no new predict step equations have to be derived.

Recall from (3.21) that the PDF of a single (Bernoulli-distributed) binary observation n_k is

$$p(n_k|x_k) = p_k^{n_k}(1-p_k)^{1-n_k}. \tag{7.1}$$

This same density function can be written as

$$p(n_k|x_k) = \begin{cases} 1-p_k & \text{if } n_k = 0 \\ p_k & \text{if } n_k = 1. \end{cases} \tag{7.2}$$

In reality, we could derive our state estimation step update equations based on (7.2) as well. For instance, if we observed $n_k = 0$ at time index k, the posterior density would be

$$p(x_k|n_{1:k}) \propto e^{n_k \log(p_k)+(1-n_k)\log(1-p_k)} \times e^{\frac{-(x_k-x_{k|k-1})^2}{2\sigma^2_{k|k-1}}} = e^{\log(1-p_k)} \times e^{\frac{-(x_k-x_{k|k-1})^2}{2\sigma^2_{k|k-1}}}, \tag{7.3}$$

where we have substituted $n_k = 0$ into the exponent of the first term. We could next take the first and second derivatives of the exponent to obtain the corresponding state estimation step update equations for $x_{k|k}$ and $\sigma^2_{k|k}$. We could also do the same for $n_k = 1$. In the case of $n_k = 1$, we would have

$$p(x_k|n_{1:k}) \propto e^{n_k \log(p_k)+(1-n_k)\log(1-p_k)} \times e^{\frac{-(x_k - x_{k|k-1})^2}{2\sigma^2_{k|k-1}}} = e^{\log(p_k)} \times e^{\frac{-(x_k - x_{k|k-1})^2}{2\sigma^2_{k|k-1}}}. \tag{7.4}$$

Finally, we could express the update step equations for the two different cases based on an *if-else* condition. This would be of the form

if $n_k = 0$,

$$x_{k|k} = \ldots \tag{7.5}$$

$$\sigma^2_{k|k} = \ldots \tag{7.6}$$

if $n_k = 1$,

$$x_{k|k} = \ldots \tag{7.7}$$

$$\sigma^2_{k|k} = \ldots. \tag{7.8}$$

In the case of an MPP where we have non-zero amplitudes *only* at the instances where point process events occur, the density function for the observations is

$$p(n_k \cap r_k|x_k) = \begin{cases} 1 - p_k = e^{\log(1-p_k)} & \text{if } n_k = 0 \\ p_k \frac{1}{\sqrt{2\pi\sigma_v^2}} e^{\frac{-(r_k - \gamma_0 - \gamma_1 x_k)^2}{2\sigma_v^2}} = e^{\log(p_k)} \frac{1}{\sqrt{2\pi\sigma_v^2}} e^{\frac{-(r_k - \gamma_0 - \gamma_1 x_k)^2}{2\sigma_v^2}} & \text{if } n_k = 1, \end{cases} \tag{7.9}$$

where the point process event amplitudes (i.e., the marks) r_k are assumed to be linearly related to x_k through $r_k = \gamma_0 + \gamma_1 x_k + v_k$, where $v_k \sim \mathcal{N}(0, \sigma_v^2)$ is sensor noise.

Let us now proceed with calculating the update step equations for the two cases where $n_k = 0$ and $n_k = 1$. First consider $n_k = 0$. Based on (7.9), the posterior density is

$$p(x_k|y_{1:k}) \propto p(n_k \cap r_k|x_k) p(x_k|n_{1:k-1}, r_{1:k-1}) \propto e^{\log(1-p_k)} \times e^{\frac{-(x_k - x_{k|k-1})^2}{2\sigma^2_{k|k-1}}}. \tag{7.10}$$

We can now take the log, take its derivative, and set it to 0 to solve for the mean. This yields

$$\frac{dq_1}{dx_k} = -\frac{1}{(1-p_k)} p_k(1-p_k) - \frac{2(x_k - x_{k|k-1})}{2\sigma^2_{k|k-1}} = 0 \tag{7.11}$$

$$\implies \frac{(x_k - x_{k|k-1})}{\sigma^2_{k|k-1}} = -p_k \tag{7.12}$$

$$x_k = x_{k|k-1} + \sigma^2_{k|k-1}(-p_k) \tag{7.13}$$

$$x_k = x_{k|k-1} + \sigma^2_{k|k-1}(n_k - p_k) \;\; \text{since } n_k = 0. \tag{7.14}$$

Interestingly, this is the same as (3.38) where we only had one binary observation n_k in the state-space model. Let us now calculate the variance by taking the second derivative.

$$\frac{d^2 q_1}{dx_k^2} = \frac{-1}{\sigma^2_{k|k-1}} - p_k(1-p_k). \tag{7.15}$$

Again, interestingly, this turns out to be the same as (3.40) where we only had one binary observation. Therefore, when a point process event does not occur (i.e., when $n_k = 0$), our state estimation step update equations are similar to the case where we only had one binary observation in the state-space model.

We will next consider the case when $n_k = 1$. Note that we will then have the r_k amplitude term as well. Based on (7.9), the posterior is now

$$p(x_k|y_{1:k}) \propto e^{\log(p_k)} \times e^{\frac{-(r_k - \gamma_0 - \gamma_1 x_k)^2}{2\sigma_v^2}} \times e^{\frac{-(x_k - x_{k|k-1})^2}{2\sigma^2_{k|k-1}}}. \tag{7.16}$$

Taking the log and proceeding to take the first derivative, we have

$$\frac{dq_2}{dx_k} = \frac{1}{p_k} p_k(1-p_k) + \frac{\gamma_1(r_k - \gamma_0 - \gamma_1 x_k)}{\sigma_v^2} - \frac{(x_k - x_{k|k-1})}{\sigma^2_{k|k-1}} = 0. \tag{7.17}$$

Since $n_k = 1$, we will replace $(1 - p_k)$ with $(n_k - p_k)$. Therefore,

$$\frac{dq_2}{dx_k} = (n_k - p_k) + \frac{\gamma_1(r_k - \gamma_0 - \gamma_1 x_k)}{\sigma_v^2} - \frac{(x_k - x_{k|k-1})}{\sigma^2_{k|k-1}} = 0. \tag{7.18}$$

This is the same as (4.21) where we had both a binary variable and a continuous variable in the state-space model. Therefore, based on (4.26), the mean update for x_k is

$$x_k = x_{k|k-1} + \frac{\sigma^2_{k|k-1}}{\gamma_1^2 \sigma^2_{k|k-1} + \sigma_v^2} \Big[\sigma_v^2 (n_k - p_k) + \gamma_1 (r_k - \gamma_0 - \gamma_1 x_{k|k-1}) \Big]. \tag{7.19}$$

Also, when we take the second derivative, we end up with

$$\frac{d^2 q_1}{dx_k^2} = -p_k(1-p_k) - \frac{\gamma_1^2}{\sigma_v^2} - \frac{1}{\sigma_{k|k-1}^2} \tag{7.20}$$

just like (4.28).

This provides an interesting insight. In the case of an MPP, the state estimation step update equations switch between those where one binary variable was observed and where both a binary variable and a continuous variable were observed. This switching occurs depending on whether $n_k = 0$ or $n_k = 1$.

When x_k gives rise to MPP observations comprising of the pairs (n_k, r_k), the update equations in the state estimation step are

if $n_k = 0$,

$$x_{k|k} = x_{k|k-1} + \sigma_{k|k-1}^2 (n_k - p_{k|k}) \tag{7.21}$$

$$\sigma_{k|k}^2 = \left[\frac{1}{\sigma_{k|k-1}^2} + p_{k|k}(1-p_{k|k}) \right]^{-1} \tag{7.22}$$

if $n_k = 1$,

$$x_{k|k} = x_{k|k-1} + \frac{\sigma_{k|k-1}^2}{\gamma_1^2 \sigma_{k|k-1}^2 + \sigma_v^2} \left[\sigma_v^2 (n_k - p_{k|k}) + \gamma_1 (r_k - \gamma_0 - \gamma_1 x_{k|k-1}) \right] \tag{7.23}$$

$$\sigma_{k|k}^2 = \left[\frac{1}{\sigma_{k|k-1}^2} + p_{k|k}(1-p_{k|k}) + \frac{\gamma_1^2}{\sigma_v^2} \right]^{-1}. \tag{7.24}$$

7.2 Deriving the Parameter Estimation Step Equations

The only changes that occur at the parameter estimation step relate to γ_0, γ_1, and σ_v^2. Parameter estimates for other variables such as the process noise variance σ_ε^2 do not change.

7.2.1 Deriving the Constant Coefficient Terms

Recall from (4.43) that when we observed one binary variable and one continuous variable, the probability term containing γ_0, γ_1, and σ_v^2 required at the parameter estimation step was

$$p(r_{1:K}|x_{1:K}, \Theta) = \prod_{k=1}^{K} \frac{1}{\sqrt{2\pi\sigma_v^2}} e^{\frac{-(r_k-\gamma_0-\gamma_1 x_k)^2}{2\sigma_v^2}}. \tag{7.25}$$

Notice that the product is over $k = 1, 2, \ldots, K$. This is when we observed a non-zero r_k *at each point in time*. When we observe an MPP variable as modeled in (7.9), r_k shows up *only* at the time indices where $n_k = 1$. Let us assume that the point process events occur at time indices $\tilde{K} \subseteq \{1, 2, \ldots, K\}$. Therefore, in the case of an MPP, the probability term we are interested in at the M-step will be

$$\prod_{k\in\tilde{K}} \frac{1}{\sqrt{2\pi\sigma_v^2}} e^{\frac{-(r_k-\gamma_0-\gamma_1 x_k)^2}{2\sigma_v^2}}, \tag{7.26}$$

where the product is only over the *specific indices* $\tilde{K}$ rather than everywhere. The corresponding log-likelihood term is therefore

$$Q = \frac{-|\tilde{K}|}{2} \log\left(2\pi\sigma_v^2\right) - \sum_{k\in\tilde{K}} \frac{\mathbb{E}\left[(r_k - \gamma_0 - \gamma_1 x_k)^2\right]}{2\sigma_v^2}. \tag{7.27}$$

We can now proceed by taking the partial derivatives with respect to γ_0, γ_1, and σ_v^2, setting them to 0 and solving. This yields

$$\sum_{k\in\tilde{K}} r_k = \gamma_0|\tilde{K}| + \gamma_1 \sum_{k\in\tilde{K}} x_{k|K} \tag{7.28}$$

$$\sum_{k\in\tilde{K}} r_k x_{k|K} = \gamma_0 \sum_{k\in\tilde{K}} x_{k|K} + \gamma_1 \sum_{k\in\tilde{K}} U_k \tag{7.29}$$

$$\begin{aligned}\sigma_v^2 = \frac{1}{|\tilde{K}|}\Bigg\{ &\sum_{k\in\tilde{K}} r_k^2 + |\tilde{K}|\gamma_0^2 + \gamma_1^2 \sum_{k\in\tilde{K}} U_k - 2\gamma_0 \sum_{k\in\tilde{K}} r_k \\ &- 2\gamma_1 \sum_{k\in\tilde{K}} r_k x_{k|K} + 2\gamma_0\gamma_1 \sum_{k\in\tilde{K}} x_{k|K} \Bigg\}.\end{aligned} \tag{7.30}$$

Note that all three equations shown above are similar to the case where a continuous variable was always present. Now, however, the summations are only over $\tilde{K}$. Thus

the parameter estimation step updates for γ_0, γ_1, and σ_v^2 are very similar to what we have seen before.

The parameter estimation step updates for γ_0, γ_1 and σ_v^2 when we observe an MPP variable with the amplitudes modeled as $r_k = \gamma_0 + \gamma_1 x_k + v_k$ are

$$\begin{bmatrix} \gamma_0 \\ \gamma_1 \end{bmatrix} = \begin{bmatrix} |\tilde{K}| & \sum_{k\in\tilde{K}} x_{k|K} \\ \sum_{k\in\tilde{K}} x_{k|K} & \sum_{k\in\tilde{K}} U_k \end{bmatrix}^{-1} \begin{bmatrix} \sum_{k\in\tilde{K}} r_k \\ \sum_{k\in\tilde{K}} r_k x_{k|K} \end{bmatrix} \tag{7.31}$$

$$\sigma_v^2 = \frac{1}{|\tilde{K}|}\Bigg\{ \sum_{k\in\tilde{K}} r_k^2 + |\tilde{K}|\gamma_0^2 + \gamma_1^2 \sum_{k\in\tilde{K}} U_k - 2\gamma_0 \sum_{k\in\tilde{K}} r_k - 2\gamma_1 \sum_{k\in\tilde{K}} r_k x_{k|K} + 2\gamma_0\gamma_1 \sum_{k\in\tilde{K}} x_{k|K} \Bigg\}. \tag{7.32}$$

7.3 MATLAB Examples

The MATLAB code examples for estimating x_k from a set of MPP observations are provided in the following folders:

- one_mpp

  sim\

  data_one_mpp.mat
  filter_one_mpp.m

  expm\

  expm_data_one_mpp.mat
  expm_filter_one_mpp.m

The code examples estimate x_k based on the inputs n_k and r_k denoted by the variables `n` and `r`. A few differences are to be noted in this code compared to the previous examples. In the previous MATLAB examples, we had the predict, update, predict, update, etc. steps executed repeatedly for $k = 1, 2, \ldots, K$. However, when we have MPP observations, we have *two* different filter update equations depending on the value of n_k. The r_k amplitudes are only taken into account when $n_k = 1$. Therefore, the state estimation step contains the following:

```
x_updt(k) = get_posterior_mode(x_pred(k), v_pred(k), r(k), r0(m),
     r1(m), vr(m), b0, n(k));
p_updt(k) = 1 / (1 + exp((-1) * (b0 + x_updt(k))));

if (n(k) == 0)
  v_updt(k) = 1 / ((1 / v_pred(k)) + p_updt(k) * (1 - p_updt(k)))
    ;
elseif (n(k) == 1)
  v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(m)) +
    p_updt(k) * (1 - p_updt(k)));
end
```

The state update, also based on an *if-else* depending on the value of n_k, is calculated using the `get_posterior_mode(...)` function shown below:

```
function [y] = get_posterior_mode(x_pred, v_pred, z, r0, r1, vr,
    b0, n)

    M = 100;    % maximum iterations
    y = NaN;

    it = zeros(1, M);
    f = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)
        if (n == 0)
            C = v_pred;
            f(i) = it(i) - x_pred - C * (n - exp(b0 + it(i)) / (1
     + exp(b0 + it(i))));
            df(i) = 1 + C * exp(b0 + it(i)) / (1 + exp(b0 + it(i)
    )) ^ 2;
        elseif (n == 1)
            C = v_pred / ((r1 ^ 2) * v_pred + vr);
            f(i) = it(i) - x_pred - C * (r1 * (z - r0 - r1 *
    x_pred) + vr * (n - (1 / (1 + exp((-1) * (b0 + it(i)))))));
            df(i) = 1 + C * vr * exp(b0 + it(i)) / ((1 + exp(b0 +
     it(i))) ^ 2);
        end

        it(i + 1)  = it(i) - f(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
            y = it(i + 1);
            return
        end
    end

    error('Newton-Raphson failed to converge.');

end
```

The other variables used in the code largely remain the same. For instance, we still use `x_pred`, `x_updt`, and `x_smth` to denote $x_{k|k-1}$, $x_{k|k}$, and $x_{k|K}$, respectively, and `v_pred`, `v_updt` and `v_smth` to denote the corresponding variances $\sigma^2_{k|k-1}$, $\sigma^2_{k|k}$ and $\sigma^2_{k|K}$.

7.3.1 Application to Skin Conductance and Sympathetic Arousal

As stated earlier, the sequence of neural impulses underlying the phasic variations in a skin conductance signal forms an MPP. This sequence of impulses is extracted via deconvolution. In the code example, the input (i.e., the deconvolved neural impulse train) is provided through the variables `n` and `r`. The variable `r(k)` has a non-zero amplitude whenever `n(k)` is equal to 1. The `r(k)` amplitudes are not taken into account when `n(k)` is 0. Running the MATLAB examples on simulated and experimental data produces the results shown in Fig. 7.1. The filter was used in [32] for estimating sympathetic arousal from deconvolved skin conductance data. The

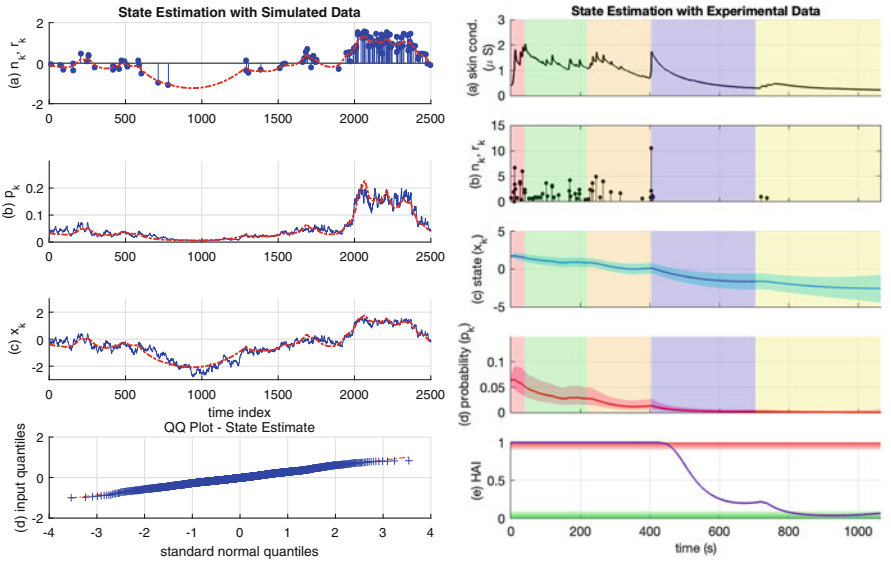

Fig. 7.1 State estimation based on observing one MPP variable. The left sub-figure depicts the estimation on simulated data and the right sub-figure depicts the estimation of sympathetic arousal from skin conductance data. The sub-panels on the left, respectively, depict (**a**) the MPP observations (blue) and the estimated r_k (red), (**b**) the point process event occurrence probability p_k (blue) and its estimate (red), (**c**) the state x_k (blue) and its estimate (red), and (**d**) the QQ plot for the residual error of x_k. The sub-panels on the right, respectively, depict (**a**) the skin conductance signal, (**b**) the neural impulses underlying phasic variations, (**c**) the arousal state x_k and its 95% confidence limits, (**d**) the probability of impulse occurrence p_k and its 95% confidence limits, and (**e**) the HAI (the regions above 90% and below 10% are shaded in red and green, respectively). The background colors on the right sub-figure correspond to the instruction period, a counting task, a color-word association task, relaxation, and watching a horror movie clip. From [32], used under Creative Commons CC-BY license

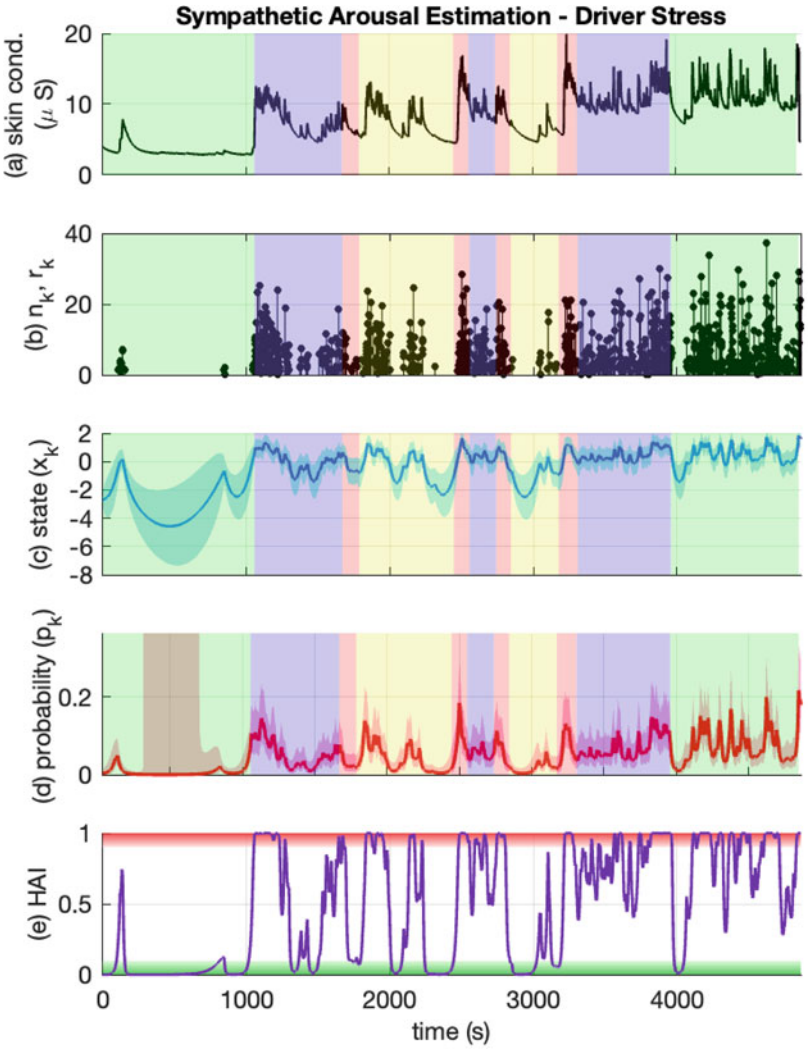

Fig. 7.2 Driver stress estimation. The sub-panels, respectively, depict (**a**) the skin conductance signal, (**b**) the neural impulses, (**c**) the arousal state x_k and its 95% confidence limits, (**d**) the probability of impulse occurrence and its 95% confidence limits, and (**e**) the HAI (the regions above 90% and below 10% are shaded in red and green, respectively). The background colors in turn denote rest, city driving, toll road, highway, toll road, city driving, toll road, highway, toll road, city driving, and rest. From [32], used under Creative Commons CC-BY license

results on experimental data shown in the figure are based on the study described in [53] (seen in the earlier chapters as well). The study involved different types of stressors interspersed by periods of relaxation. The results of using the same code on the driver stress data in [54] for a particular subject are shown in Fig. 7.2.

Chapter 8
State-Space Model with One MPP and One Continuous Observation

In this chapter, we will derive the EM algorithm equations for a state-space model having an MPP and a continuous-valued variable as its observations. Before looking at the state-space model itself and the equation derivations, we will again first consider a scenario for where the need for such a model arises. We stated earlier that the human body is comprised of multiple internal sub-systems that are networked with one another. The sub-systems perform specialized functions and all work in unison to maintain life. Now multiple functions within the body are regulated by the endocrine system. The endocrine system governs the secretion of a number of hormones that act on different target cells in the body. These hormones largely serve as messengers and help coordinate activities between sub-systems within the body. Functions that hormones are involved in include metabolism, the regulation of blood glucose and appetite, and playing a role in the body's immune and stress responses, to name a few [101].

The secretory mechanism is pulsatile in the case of a number of hormones. Cortisol is one such example [38]. One of the major functions of cortisol is to raise blood glucose levels in response to external stressors [102, 103]. When the brain interprets sensory inputs as requiring cortisol secretion, the hypothalamus begins to secrete the hormone CRH (corticotropin-releasing hormone). CRH in turn causes the secretion of ACTH (adrenocorticotropic hormone) from the anterior pituitary. Finally, ACTH causes the secretion of cortisol from the adrenal glands [104]. The secretion of cortisol from the adrenal glands has a negative feedback effect suppressing the further secretion of CRH and ACTH [105, 106]. Between 15 and 22 cortisol secretory events typically occur each day in a healthy adult [38, 107]. When cortisol is secreted into the bloodstream, a large percentage of it remains

Supplementary Information The online version contains supplementary material available at (https://doi.org/10.1007/978-3-031-47104-9_8).

GitHub https://github.com/computational-medicine-lab/A-Tutorial-on-the-Design-of-Bayesian-Filters/tree/main/one_mpp_one_cont

D. S. Wickramasuriya, R. T. Faghih, *Bayesian Filter Design for Computational Medicine*, https://doi.org/10.1007/978-3-031-47104-9_8

bound [108]. It is the unbound cortisol in the blood that remains physiologically active [109]. This active cortisol aids in energy production at the liver [110, 111]. Since the cortisol concentration in the blood is a continuous variable and its pulsatile secretion forms an MPP, a state-space model for estimating the energy production level related to it should incorporate these types of observations. Similar to the case of skin conductance, a deconvolution procedure can be used to extract the pulsatile profile underlying a series of blood cortisol measurements [107]. Deconvolution also typically yields the infusion and clearance rates necessary to reconstruct a minute-by-minute profile of the cortisol concentration in the blood. Figure 8.1 shows a deconvolved cortisol profile [113].

Alternately, the same MPP plus continuous variable formulation can also be applied to skin conductance. Recall that skin conductance contains both a fast-varying phasic component and a slow-varying tonic component. The phasic component consists of a series of SCRs that are generated by neural impulses. These neural impulses form an MPP. The tonic component, which also reflects sympathetic arousal information, is a continuous observation [60]. Consequently, the state-space model with an MPP and a continuous observation can also be applied to the case of skin conductance. Unlike the case where we had one binary observation and two continuous observations to estimate sympathetic arousal from the same information, the formulation with the MPP and the continuous observation conforms more intuitively to the data itself.

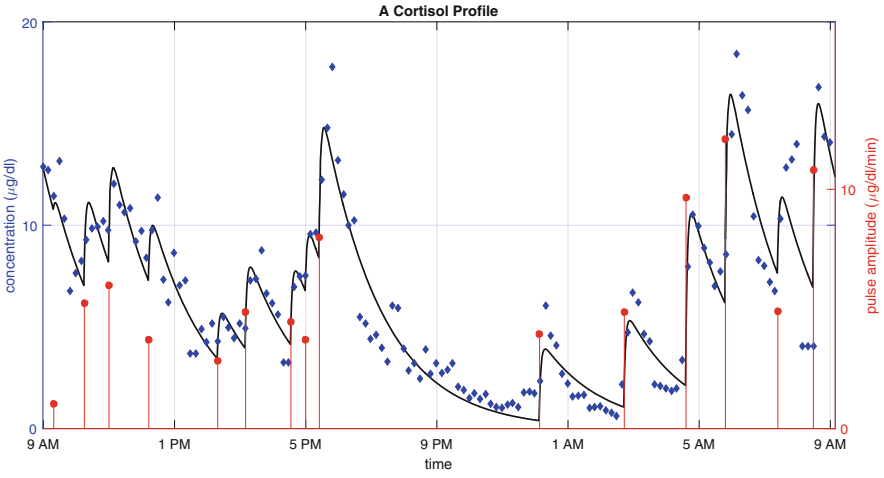

Fig. 8.1 A deconvolved cortisol profile. Cortisol is secreted in pulses and between 15 and 22 secretory events occur each day in a healthy adult. The figure depicts the blood cortisol measurements taken at 10 min intervals (blue), the reconstructed blood cortisol concentrations at a 1 min resolution (black), and the pulsatile secretions (red). From [112], used under Creative Commons CC-BY license

8.1 Deriving the Update Equations in the State Estimation Step

Here again we will assume that x_k varies with time following one of the state equations we have already seen. Therefore, no new predict step equations need to be derived.

We made an interesting observation in the previous chapter when deriving the update step equations for the case where x_k gives rise to MPP observations. We observed that the update equations switched between those where one binary variable was observed and where both a binary variable and a continuous variable were observed. We will now consider the case where we observe an MPP variable along with a continuous variable. As in (7.9), the density function for our MPP is still

$$p(n_k \cap r_k|x_k) = \begin{cases} 1 - p_k = e^{\log(1-p_k)} & \text{if } n_k = 0 \\ p_k \frac{1}{\sqrt{2\pi\sigma_v^2}} e^{\frac{-(r_k-\gamma_0-\gamma_1 x_k)^2}{2\sigma_v^2}} = e^{\log(p_k)} \frac{1}{\sqrt{2\pi\sigma_v^2}} e^{\frac{-(r_k-\gamma_0-\gamma_1 x_k)^2}{2\sigma_v^2}} & \text{if } n_k = 1, \end{cases} \tag{8.1}$$

where n_k and r_k denote the occurrence of the point process events and the mark amplitudes, respectively. In addition to the MPP, we will now assume that we also observe a continuous variable s_k where

$$s_k = \delta_0 + \delta_1 x_k + w_k, \tag{8.2}$$

and δ_0, δ_1, and w_k have their usual meanings. We observe s_k at every point in time. Let us now proceed with deriving the mean and variance for the case when $n_k = 0$. The posterior density in this case is

$$p(x_k|y_{1:k}) \propto e^{\log(1-p_k)} \times e^{\frac{-(s_k-\delta_0-\delta_1 x_k)^2}{2\sigma_w^2}} \times e^{\frac{-(x_k-x_{k|k-1})^2}{2\sigma_{k|k-1}^2}}. \tag{8.3}$$

Taking the log on both sides, we have

$$q_1 = \log(1 - p_k) - \frac{(s_k - \delta_0 - \delta_1 x_k)^2}{2\sigma_w^2} - \frac{(x_k - x_{k|k-1})^2}{2\sigma_{k|k-1}^2} + \text{constant}. \tag{8.4}$$

We will next take the first and second derivatives of q_1 to obtain the mean and variance.

$$\frac{dq_1}{dx_k} = -\frac{1}{(1-p_k)} p_k(1-p_k) + \frac{\delta_1(s_k - \delta_0 - \delta_1 x_k)}{\sigma_w^2} - \frac{(x_k - x_{k|k-1})}{\sigma_{k|k-1}^2} = 0 \tag{8.5}$$

$$\implies 0 = -p_k + \frac{\delta_1(s_k - \delta_0 - \delta_1 x_k)}{\sigma_w^2} - \frac{(x_k - x_{k|k-1})}{\sigma_{k|k-1}^2}. \tag{8.6}$$

Since $n_k = 0$, we can rewrite $-p_k$ as $(n_k - p_k)$. Therefore,

$$n_k - p_k + \frac{\delta_1(s_k - \delta_0 - \delta_1 x_k)}{\sigma_w^2} = \frac{(x_k - x_{k|k-1})}{\sigma_{k|k-1}^2}. \tag{8.7}$$

But this is identical to (4.21) with s_k, δ_0, and δ_1 appearing in the equation instead of r_k, γ_0, and γ_1. Therefore, similar to (4.26), the update for the mean is

$$x_k = x_{k|k-1} + \frac{\sigma_{k|k-1}^2}{\delta_1^2\sigma_{k|k-1}^2 + \sigma_w^2}\Big[\sigma_w^2(n_k - p_k) + \delta_1(s_k - \delta_0 - \delta_1 x_{k|k-1})\Big]. \tag{8.8}$$

We next take the second derivative of q_1.

$$\frac{d^2 q_1}{dx_k^2} = -p_k(1 - p_k) - \frac{\delta_1^2}{\sigma_w^2} - \frac{1}{\sigma_{k|k-1}^2}. \tag{8.9}$$

This also happens to be identical to (4.28) but with δ_1 and σ_w^2 instead of γ_1 and σ_v^2. Therefore, similar to (4.29), the variance update is

$$\sigma_{k|k}^2 = -\left(\frac{d^2 q_1}{dx_k^2}\right)^{-1} = \left[\frac{1}{\sigma_{k|k-1}^2} + p_{k|k}(1 - p_{k|k}) + \frac{\delta_1^2}{\sigma_w^2}\right]^{-1}. \tag{8.10}$$

This is interesting. When we observe both an MPP variable and a continuous variable and $n_k = 0$, the update equations are identical to the case where one binary variable and one continuous variable were observed.

We will next consider the case where $n_k = 1$ and a non-zero mark r_k exists. In this case, the posterior density is

$$p(x_k|y_{1:k}) \propto e^{\log(p_k)} \times e^{\frac{-(r_k - \gamma_0 - \gamma_1 x_k)^2}{2\sigma_v^2}} \times e^{\frac{-(s_k - \delta_0 - \delta_1 x_k)^2}{2\sigma_w^2}} \times e^{\frac{-(x_k - x_{k|k-1})^2}{2\sigma_{k|k-1}^2}}. \tag{8.11}$$

Taking the log value and proceeding to take the first derivative, we have

$$\frac{dq_2}{dx_k} = \frac{1}{p_k} p_k(1 - p_k) + \frac{\gamma_1(r_k - \gamma_0 - \gamma_1 x_k)}{\sigma_v^2} + \frac{\delta_1(s_k - \delta_0 - \delta_1 x_k)}{\sigma_w^2} - \frac{(x_k - x_{k|k-1})}{\sigma_{k|k-1}^2} \tag{8.12}$$

$$= 1 - p_k + \frac{\gamma_1(r_k - \gamma_0 - \gamma_1 x_k)}{\sigma_v^2} + \frac{\delta_1(s_k - \delta_0 - \delta_1 x_k)}{\sigma_w^2} - \frac{(x_k - x_{k|k-1})}{\sigma_{k|k-1}^2}. \tag{8.13}$$

Setting this to 0 and replacing $(1 - p_k)$ with $(n_k - p_k)$ since $n_k = 1$, we have

$$\frac{dq_2}{dx_k} = n_k - p_k + \frac{\gamma_1(r_k - \gamma_0 - \gamma_1 x_k)}{\sigma_v^2} + \frac{\delta_1(s_k - \delta_0 - \delta_1 x_k)}{\sigma_w^2} - \frac{(x_k - x_{k|k-1})}{\sigma_{k|k-1}^2} = 0. \tag{8.14}$$

But this is identical to (5.21) where we observed one binary variable and two continuous variables. The second derivative of q_2 yields

$$\frac{d^2 q_2}{dx_k^2} = -p_k(1 - p_k) - \frac{\gamma_1^2}{\sigma_v^2} - \frac{\delta_1^2}{\sigma_w^2} - \frac{1}{\sigma_{k|k-1}^2}, \tag{8.15}$$

which is the same as (5.28). Therefore, in the case where we observe an MPP variable along with a continuous variable and $n_k = 1$, our update equations in the state estimation step are identical to those where we have one binary variable and two continuous variables.

When x_k gives rise to MPP observations comprising of the pairs (n_k, r_k) and a continuous observation s_k, the update equations in the state estimation step are

if $n_k = 0$,

$$x_k = x_{k|k-1} + \frac{\sigma_{k|k-1}^2}{\gamma_1^2 \sigma_{k|k-1}^2 + \sigma_w^2}\left[\sigma_w^2(n_k - p_k) + \gamma_1(s_k - \delta_0 - \delta_1 x_{k|k-1})\right] \tag{8.16}$$

$$\sigma_{k|k}^2 = \left[\frac{1}{\sigma_{k|k-1}^2} + p_{k|k}(1 - p_{k|k}) + \frac{\delta_1^2}{\sigma_w^2}\right]^{-1} \tag{8.17}$$

if $n_k = 1$,

$$x_{k|k} = x_{k|k-1} + \frac{\sigma_{k|k-1}^2}{\sigma_v^2\sigma_w^2 + \sigma_{k|k-1}^2(\gamma_1^2\sigma_w^2 + \delta_1^2\sigma_v^2)}\Big[\sigma_v^2\sigma_w^2(n_k - p_{k|k}) + \gamma_1\sigma_w^2(r_k - \gamma_0 - \gamma_1 x_{k|k-1}) + \delta_1\sigma_v^2(s_k - \delta_0 - \delta_1 x_{k|k-1})\Big] \tag{8.18}$$

$$\sigma_{k|k}^2 = \left[\frac{1}{\sigma_{k|k-1}^2} + p_{k|k}(1 - p_{k|k}) + \frac{\gamma_1^2}{\sigma_v^2} + \frac{\delta_1^2}{\sigma_w^2}\right]^{-1}. \tag{8.19}$$

8.2 Deriving the Parameter Estimation Step Equations

The derivation of the parameter estimation step updates is similar to what we have seen thus far. The updates for the parameters γ_0, γ_1, and σ_v^2 related to r_k are calculated based on the subset of values $\tilde{K}$ corresponding to where $n_k = 1$. The parameters δ_0, δ_1, and σ_w^2 corresponding to s_k are calculated based on all the observations.

8.3 MATLAB Examples

The MATLAB code examples are contained in the folders shown below:

- one_mpp_one_cont

  sim\

  data_one_mpp_one_cont.mat
  filter_one_mpp_one_cont.m

  expm\

  expm_data_one_mpp_one_cont.mat
  expm_filter_one_mpp_one_cont.m

The code itself is quite similar to what we have seen before in earlier examples. It takes in the inputs n_k, r_k, and s_k denoted by the variables `n`, `r`, and `s` to estimate x_k. We progress through the repeated predict, update, predict, update, etc. steps with $x_{k|k}$ and $\sigma_{k|k}^2$ being estimated using different equations based on n_k. The variable names are also largely similar to what we have seen earlier.

8.3.1 Application to Cortisol and Energy

Recall the discussion regarding cortisol at the beginning of this chapter. Cortisol is secreted in pulses and between 15 and 22 of them are secreted by a healthy adult each day. The pulsatile hormone profile forms an MPP. In addition, the amount of unbound cortisol in the blood is biologically active and contributes to energy production. Thus, the observations for estimating the latent cortisol-related energy production state form an MPP and a continuous-valued variable. The cortisol inputs are provided to the code using the variables `n`, `r`, and `s`. The variables `n` and `r` denote the MPP observations n_k and r_k. The pulsatile secretions forming the MPP at a resolution of 1 min will need to be extracted via deconvolution (e.g., using [107, 113, 114]). The cortisol infusion and clearance rates yielded by the deconvolution algorithm are used to generate s_k.

Running the code examples for this particular state-space model produces the results in Fig. 8.2. The code running on experimental data for this model contains a notable difference. In general, when a continuous-valued observation is present, the state estimate can tend to overfit to it. In the experimental code example, the parameter estimation step updates for δ_0, δ_1, and σ_w^2 (the three parameters related to s_k) have been adjusted so that only a small step is taken in the direction of the next predicted values at a time. A second change has also been made in that the sensor noise variance σ_w^2 is initialized at a larger value and the same three parameters δ_0, δ_1, and σ_w^2 are only permitted to update until σ_w^2 reaches a threshold. These two changes greatly help reduce the overfitting to s_k.

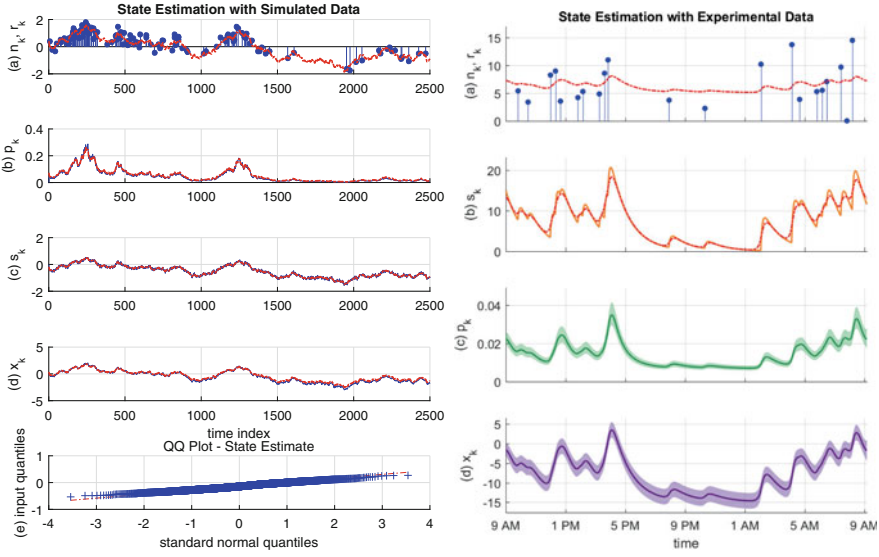

Fig. 8.2 State estimation based on observing one MPP and one continuous variable. The left sub-figure depicts the estimation on simulated data and the right sub-figure depicts the estimation of energy from blood cortisol data. The sub-panels on the left, respectively, depict (**a**) the MPP observations (blue) and the estimated r_k (red), (**b**) the point process event occurrence probability p_k (blue) and its estimate (red), (**c**) the continuous-valued variable s_k (blue) and its estimate (red), (**d**) the state x_k (blue) and its estimate (red), and (**e**) the QQ plot for the residual error of x_k. The sub-panels on the right, respectively, depict (**a**) the deconvolved cortisol pulses (blue) and the fit to r_k (red), (**b**) the reconstructed blood cortisol profile s_k (orange) and its estimate (red), (**c**) the probability of pulse occurrence p_k and its 95% confidence limits, and (**d**) the energy state x_k and its 95% confidence limits. From [33], used under Creative Commons CC-BY license

Chapter 9
Additional Models and Derivations

Much of what we have described in the preceding chapters provides the basic tools necessary to build physiological state-space estimators. In this chapter, we will briefly review some additional concepts in state-space estimation, a non-traditional method of estimation, and some supplementary models. These may help serve as pointers if extensions are to be built to the models already described.

9.1 State-Space Model with a Time-Varying Process Noise Variance Based on a GARCH(p, q) Framework

Thus far, we have not considered time-varying model parameters. In reality, the human body is not static. Instead it undergoes changes from time to time (e.g., due to disease conditions, adaptation to new environments). In this section, we will consider a state equation of the form

$$x_k = x_{k-1} + \varepsilon_k \tag{9.1}$$

where $\varepsilon_k \sim \mathcal{N}(0, \sigma^2_{\varepsilon,k})$. Note that the process noise variance now depends on the time index k. Here we will use concepts from the Generalized Autoregressive Conditional Heteroskedasticity (GARCH) framework to model ε_k. In a general GARCH(p, q) framework, we take

$$\varepsilon_k = h_k \nu_k, \tag{9.2}$$

Supplementary Information The online version contains supplementary material available at (https://doi.org/10.1007/978-3-031-47104-9_9).

GitHub https://github.com/computational-medicine-lab/A-Tutorial-on-the-Design-of-Bayesian-Filters/tree/main/one_bin_one_spk

GitHub https://github.com/computational-medicine-lab/A-Tutorial-on-the-Design-of-Bayesian-Filters/tree/main/one_bin_two_cont_circadian

D. S. Wickramasuriya, R. T. Faghih, *Bayesian Filter Design for Computational Medicine*, https://doi.org/10.1007/978-3-031-47104-9_9

where $\nu_k \sim \mathcal{N}(0, 1)$ and

$$h_k^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{k-i}^2 + \sum_{j=1}^{p} \beta_j h_{k-j}^2, \tag{9.3}$$

where the α_i's and β_j's are coefficients to be determined. Now, conditioned on having observed all the sensor readings up to time index $(k-1)$, we have

$$\mathbb{E}[\varepsilon_k] = \mathbb{E}[h_k \nu_k] = h_k \mathbb{E}[\nu_k] = h_k \times 0 = 0 \tag{9.4}$$

and

$$\sigma_{\varepsilon,k}^2 = V(\varepsilon_k) = V(h_k \nu_k) = h_k^2 V(\nu_k) = h_k^2 \times 1 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{k-i}^2 + \sum_{j=1}^{p} \beta_j h_{k-j}^2. \tag{9.5}$$

As is evident from (9.5), the variance of ε_k depends on k. If a GARCH(p, q) model is used for the process noise term in the random walk, the predict equations in the state estimation step change to

$$x_{k|k-1} = x_{k-1|k-1} \tag{9.6}$$

$$\sigma_{k|k-1}^2 = \sigma_{k-1|k-1}^2 + \sigma_{\varepsilon,k}^2 = \sigma_{k-1|k-1}^2 + \alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{k-i}^2 + \sum_{j=1}^{p} \beta_j h_{k-j}^2. \tag{9.7}$$

The update equations in the state estimation step remain unchanged. Note also that the calculation of $\sigma_{k|k-1}^2$ requires the previous process noise terms. In general, these will have to be calculated based on successive differences between the x_k and x_{k-1} estimates.

Moreover, we would also have $(p + q + 1)$ additional GARCH terms (the α_i's and β_j's) to determine at the parameter estimation step. These terms would have to be chosen to maximize the log-likelihood

$$Q = \frac{(-1)}{2} \sum_{k=1}^{K} \mathbb{E}\left[\log(2\pi\sigma_{\varepsilon,k}^2) + \frac{(x_k - x_{k-1})^2}{\sigma_{\varepsilon,k}^2} \right] \tag{9.8}$$

$$= \frac{(-1)}{2} \sum_{k=1}^{K} \mathbb{E}\left\{ \log\left[2\pi \left(\alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{k-i}^2 + \sum_{j=1}^{p} \beta_j h_{k-j}^2 \right) \right] + \frac{(x_k - x_{k-1})^2}{\alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{k-i}^2 + \sum_{j=1}^{p} \beta_j h_{k-j}^2} \right\}. \tag{9.9}$$

The maximization of Q with respect to the GARCH terms is rather complicated. Choosing a GARCH(1, 1) model for ε_k simplifies the computations somewhat. Additionally, note the recursive form contained within Q. For each value of k, we have terms of the form h_{k-j}^2 which contain within them further h^2 terms. In general, computing Q is challenging unless further simplifying assumptions are made.

When x_k evolves with time following $x_k = x_{k-1} + \varepsilon_k$, where ε_k is modeled using a GARCH(p, q) framework, the predict equations in the state estimation step are

$$x_{k|k-1} = x_{k-1|k-1} \tag{9.10}$$

$$\sigma_{k|k-1}^2 = \sigma_{k-1|k-1}^2 + \alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{k-i}^2 + \sum_{j=1}^{p} \beta_j h_{k-j}^2. \tag{9.11}$$

The parameter estimation step updates for the $(p + q + 1)$ GARCH terms are chosen to maximize

$$\frac{(-1)}{2} \sum_{k=1}^{K} \mathbb{E}\left\{ \log\left[2\pi \left(\alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{k-i}^2 + \sum_{j=1}^{p} \beta_j h_{k-j}^2 \right) \right] + \frac{(x_k - x_{k-1})^2}{\alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{k-i}^2 + \sum_{j=1}^{p} \beta_j h_{k-j}^2} \right\}. \tag{9.12}$$

9.2 Deriving the Parameter Estimation Step Equations for Terms Related to a Binary Observation

Thus far, we have only considered cases where the probability of binary event occurrence p_k is of the form

$$p_k = \frac{1}{1 + e^{-(\beta_0 + x_k)}}. \tag{9.13}$$

We have also thus far only estimated β_0 empirically (e.g., based on the average probability of point process event occurrence). Occasionally, however, we will find it helpful to model p_k as

$$p_k = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_k)}} \tag{9.14}$$

and determine β_0 and β_1 at the parameter estimation step. If we wish to do so, we will need to consider the probability term that needs to be maximized at this step. Based on (3.27), this probability term is

$$\prod_{k=1}^{K} e^{n_k \log\left(\frac{p_k}{1-p_k}\right)+\log(1-p_k)} = \prod_{k=1}^{K} e^{n_k(\beta_0+\beta_1 x_k)+\log\left(\frac{1}{1+e^{\beta_0+\beta_1 x_k}}\right)}. \tag{9.15}$$

This yields the expected log-likelihood

$$Q = \sum_{k=1}^{K} \mathbb{E}\Big[n_k(\beta_0+\beta_1 x_k) - \log\left(1+e^{\beta_0+\beta_1 x_k}\right)\Big]. \tag{9.16}$$

As in the case of determining the parameter updates for the terms in a CIF, this expected value is also somewhat complicated. Again, the trick is to perform a Taylor expansion around the mean $\mathbb{E}[x_k] = x_{k|K}$ for each of the individual log terms. After performing this expansion, we end up with terms like $\mathbb{E}[x_k - x_{k|K}]$ and $\mathbb{E}[(x_k - x_{k|K})^2]$ which greatly simplify our calculations.

Let us begin by performing a Taylor expansion of the log term around $x_{k|K}$ [6].

$$\begin{aligned}\log\left(1+e^{\beta_0+\beta_1 x_k}\right) \approx{}& \log\left(1+e^{\beta_0+\beta_1 x_{k|K}}\right) + \beta_1 p_{k|K}(x_k - x_{k|K}) \\ &+ \frac{\beta_1^2}{2} p_{k|K}(1-p_{k|K})(x_k - x_{k|K})^2.\end{aligned} \tag{9.17}$$

Note the terms $(x_k - x_{k|K})$ and $(x_k - x_{k|K})^2$ in the expansion. Taking the expected value on both sides,

$$\begin{aligned}\mathbb{E}\Big[\log\left(1+e^{\beta_0+\beta_1 x_k}\right)\Big] \approx{}& \log\left(1+e^{\beta_0+\beta_1 x_{k|K}}\right) + \beta_1 p_{k|K}\mathbb{E}\big[x_k - x_{k|K}\big] \\ &+ \frac{\beta_1^2}{2} p_{k|K}(1-p_{k|K})\mathbb{E}\big[(x_k - x_{k|K})^2\big] \qquad (9.18)\\ ={}& \log\left(1+e^{\beta_0+\beta_1 x_{k|K}}\right) + 0 + \frac{\beta_1^2}{2} p_{k|K}(1-p_{k|K})\sigma^2_{k|K}. \qquad (9.19)\end{aligned}$$

Therefore,

$$Q \approx \sum_{k=1}^{K}\left[n_k(\beta_0+\beta_1 x_{k|K}) - \log\left(1+e^{\beta_0+\beta_1 x_{k|K}}\right) - \frac{\beta_1^2}{2} p_{k|K}(1-p_{k|K})\sigma^2_{k|K}\right]. \tag{9.20}$$

Now,

$$\frac{\partial p_{k|K}}{\partial \beta_0} = \frac{\partial}{\partial \beta_0}\left[\frac{1}{1+e^{-(\beta_0+\beta_1 x_{k|K})}}\right] = \frac{(-1)}{\left[1+e^{-(\beta_0+\beta_1 x_{k|K})}\right]^2} \times \left[-e^{-(\beta_0+\beta_1 x_{k|K})}\right]$$

$$= p_{k|K}(1-p_{k|K}). \tag{9.21}$$

And similarly,

$$\frac{\partial p_{k|K}}{\partial \beta_1} = p_{k|K}(1-p_{k|K})x_{k|K}. \tag{9.22}$$

Taking the partial derivative of Q with respect to β_0, we have

$$\frac{\partial Q}{\partial \beta_0} = \sum_{k=1}^{K}\left\{n_k - \frac{e^{\beta_0+\beta_1 x_{k|K}}}{\left(1+e^{\beta_0+\beta_1 x_{k|K}}\right)} - \frac{\beta_1^2\sigma_{k|K}^2}{2}\frac{\partial}{\partial \beta_0}\Big[p_{k|K}(1-p_{k|K})\Big]\right\} \tag{9.23}$$

$$= \sum_{k=1}^{K}\left\{n_k - p_{k|K} - \frac{\beta_1^2\sigma_{k|K}^2}{2}\frac{\partial}{\partial \beta_0}\Big[p_{k|K}(1-p_{k|K})\Big]\right\} \tag{9.24}$$

$$= \sum_{k=1}^{K}\left[n_k - p_{k|K} - \frac{\beta_1^2\sigma_{k|K}^2}{2}(1-p_{k|K})(1-2p_{k|K})p_{k|K}\right]. \tag{9.25}$$

And similarly for β_1, we have

$$\frac{\partial Q}{\partial \beta_1} = \sum_{k=1}^{K}\left[n_k x_{k|K} - x_{k|K}\,p_{k|K} - \frac{\beta_1\sigma_{k|K}^2}{2}p_{k|K}(1-p_{k|K})\big[2+\beta_1 x_{k|K}(1-2p_{k|K})\big]\right]. \tag{9.26}$$

By setting

$$\frac{\partial Q}{\partial \beta_0} = 0 \tag{9.27}$$

$$\frac{\partial Q}{\partial \beta_1} = 0, \tag{9.28}$$

we obtain two simultaneous equations with which to solve for β_0 and β_1. Note also that the use of β_0 and β_1 in p_k causes changes to the filter update equations for $x_{k|k}$ and $\sigma_{k|k}^2$.

The parameter estimation step updates for β_0 and β_1 when we observe a binary variable n_k are obtained by solving

$$\sum_{k=1}^{K}\left[n_k - p_{k|K} - \frac{\beta_1^2\sigma_{k|K}^2}{2}(1-p_{k|K})(1-2p_{k|K})p_{k|K}\right] = 0 \tag{9.29}$$

$$\sum_{k=1}^{K}\left[n_k x_{k|K} - x_{k|K}\,p_{k|K} - \frac{\beta_1\sigma_{k|K}^2}{2}p_{k|K}(1-p_{k|K})\big[2+\beta_1 x_{k|K}(1-2p_{k|K})\big]\right] = 0. \tag{9.30}$$

9.3 Extending Estimation to a Vector-Valued State

We have also thus far only considered cases where a single state x_k gives rise to different observations. In a number of applications, we will encounter the need to estimate a vector-valued state $\mathbf{x}_k$. For instance, we may need to estimate the position of a small animal on a 2D plane from neural spiking observations or may need to estimate different aspects of emotion from physiological signal features. We have a multi-dimensional $\mathbf{x}_k$ in each of these cases.

Let us first consider the predict equations in the state estimation step. Assume that we have a state $\mathbf{x}_k$ that varies with time following

$$\mathbf{x}_k = A\mathbf{x}_{k-1} + B\mathbf{u}_k + \mathbf{e}_k, \tag{9.31}$$

where A and B are matrices and $\mathbf{e}_k \sim \mathcal{N}(\mathbf{0}, \Sigma)$ is the process noise. The basic statistical results related to mean and variance in (2.1)–(2.6) simply generalize to the vector case. Thus, the predict equations in the state estimation step become

$$\mathbf{x}_{k|k-1} = A\mathbf{x}_{k-1|k-1} + B\mathbf{u}_k \tag{9.32}$$

$$\Sigma_{k|k-1} = A\,\Sigma_{k-1|k-1}A^{\mathsf{T}} + \Sigma, \tag{9.33}$$

where the covariance (uncertainty) Σ of $\mathbf{x}_k$ is now a matrix.

Recall also how we derived the update equations in the state estimation step. We calculated the terms that appeared in posterior $p(x_k|y_{1:k})$ and made a Gaussian approximation to it in order to derive the mean and variance updates $x_{k|k}$ and $\sigma_{k|k}^2$. In all of the scalar cases, the log posterior density had the form

$$q_s = f(x_k) - \frac{(x_k - x_{k|k-1})^2}{2\sigma^2_{k|k-1}} + \text{constant}, \tag{9.34}$$

where $f(x_k)$ was some function of x_k. This function could take on different forms depending on whether binary, continuous, or spiking-type observations (or different combinations of them) were present. In each of the cases, the mean and variance were derived based on the first and second derivatives of q_s.

There are two different ways for calculating the update step equations in the vector case.

- The first is the traditional approach outlined in [10]. Here, the result that holds for the 1D case is simply extended to the vector case. Regardless of the types of observations (features) that are present in the state-space model, the log posterior is of the form

$$q_v = f(\mathbf{x}_k) - \frac{1}{2}(\mathbf{x}_k - \mathbf{x}_{k|k-1})^\intercal \Sigma^{-1}_{k|k-1}(\mathbf{x}_k - \mathbf{x}_{k|k-1}) + \text{constant}. \tag{9.35}$$

  The manner in which the updates $\mathbf{x}_{k|k}$ and $\Sigma_{k|k}$ are calculated, however, is quite similar. We simply take the first *vector* derivative of q_v and solve for where it is $\mathbf{0}$ to obtain $\mathbf{x}_{k|k}$. We next take the *Hessian* of q_v comprising all the second derivatives and take its negative inverse to obtain $\Sigma_{k|k}$.
- The second approach is slightly different [115]. Note that, based on making a Gaussian approximation to the log posterior, we can write

$$-\frac{1}{2}(\mathbf{x}_k - \mathbf{x}_{k|k})^\intercal \Sigma^{-1}_{k|k}(\mathbf{x}_k - \mathbf{x}_{k|k}) = f(\mathbf{x}_k) - \frac{1}{2}(\mathbf{x}_k - \mathbf{x}_{k|k-1})^\intercal \Sigma^{-1}_{k|k-1}(\mathbf{x}_k - \mathbf{x}_{k|k-1}) + \text{constant}. \tag{9.36}$$

  Let us take the first vector derivative with respect to $\mathbf{x}_k$ on both sides. This yields

$$-\Sigma^{-1}_{k|k}(\mathbf{x}_k - \mathbf{x}_{k|k}) = \frac{\partial f(\mathbf{x}_k)}{\partial \mathbf{x}_k} - \Sigma^{-1}_{k|k-1}(\mathbf{x}_k - \mathbf{x}_{k|k-1}). \tag{9.37}$$

  Let us now *evaluate* this expression at $\mathbf{x}_k = \mathbf{x}_{k|k-1}$. Do you see that if we substitute $\mathbf{x}_k = \mathbf{x}_{k|k-1}$ in the above expression, the second term on the right *simply goes away*? Therefore, we end up with

$$-\Sigma^{-1}_{k|k}(\mathbf{x}_{k|k-1} - \mathbf{x}_{k|k}) = \left.\frac{\partial f(\mathbf{x}_k)}{\partial \mathbf{x}_k}\right|_{\mathbf{x}_{k|k-1}} \tag{9.38}$$

$$\implies \mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + \Sigma_{k|k}\left.\frac{\partial f(\mathbf{x}_k)}{\partial \mathbf{x}_k}\right|_{\mathbf{x}_{k|k-1}}. \tag{9.39}$$

This yields the mean state update for $\mathbf{x}_{k|k}$. How do we derive the covariance matrix $\Sigma_{k|k}$? We simply take the vector derivative of (9.37) again. Note that in this case, $\frac{\partial^2}{\partial \mathbf{x}_k^2}$ is a matrix of all the second derivative terms. Thus, we obtain

$$\Sigma_{k|k}^{-1} = -\frac{\partial^2 f(\mathbf{x}_k)}{\partial \mathbf{x}_k^2} + \Sigma_{k|k-1}^{-1} \tag{9.40}$$

$$\implies \Sigma_{k|k} = \left[-\frac{\partial^2 f(\mathbf{x}_k)}{\partial \mathbf{x}_k^2} + \Sigma_{k|k-1}^{-1} \right]^{-1}. \tag{9.41}$$

9.4 The Use of Machine Learning Methods for State Estimation

Machine learning approaches can also be used for state estimation (e.g., [116, 117]). In these methods, neural networks or other techniques are utilized to learn a particular state-space model and infer the unobserved state(s) from a dataset. In this section, we will briefly describe how the neural network approach in [116] is used for estimation. In [116], Krishnan et al. considered the general Gaussian state-space model

$$x_k \sim \mathcal{N}(f_{\mu_x}(x_{k-1}), f_{\sigma_x^2}(x_{k-1})) \tag{9.42}$$

$$y_k \sim \Pi(f_y(x_k)), \tag{9.43}$$

where y_k represents the observations. Both the state equation and the output equation are learned using two separate neural networks (for simplicity, we group both of them together under the title "state-space neural network"—SSNN). A separate recurrent neural network (RNN) is used to estimate x_k. Taking ψ and ϕ to denote the parameters of the state-space model and the RNN, respectively, the networks are trained by maximizing

$$\begin{aligned}\tilde{Q} = &\sum_{k=1}^{K} \mathbb{E}_{q_\phi(x_k|\vec{y})}\big[\log p_\psi(y_k|x_k)\big] - \mathrm{KL}(q_\phi(x_1|\vec{y})||p_\psi(x_1)) \\ &- \sum_{k=2}^{K} \mathbb{E}_{q_\phi(x_{k-1}|\vec{y})}\big[\mathrm{KL}(q_\phi(x_k|x_{k-1}, \vec{y})||p_\psi(x_k|x_{k-1}))\big],\end{aligned} \tag{9.44}$$

where $p_\psi(\cdot)$ and $q_\phi(\cdot)$ denote density functions [116]. The actual training is performed within the algorithm as a minimization of the negative term which we label Q_{ML}. Analogous to the state-space EM algorithms we have seen so far, in

this neural network approach, the SSNN replaces the explicit state-space model, the RNN replaces the Bayesian filter, and the weights of the neural networks replace the model parameters. The objective, however, is still to estimate x_k from observations such as n_k, r_k, and s_k. Since neural networks are used to learn the state-space model, more complicated state transitions and input-output relationships are permitted. One of the drawbacks, however, is that a certain degree of interpretability is lost.

Similarities also exist between the terms in Q_{ML} and the log-likelihood terms we have seen thus far. For instance, when a binary variable n_k is present among the observations y_k, Q_{ML} contains the summation

$$-\sum \left[n_k \log \left(\frac{1}{1+e^{-f_n(x_k)}} \right) + (1-n_k) \log \left(1 - \frac{1}{1+e^{-f_n(x_k)}} \right) \right]. \tag{9.45}$$

Take a moment to look back at how (3.15) and (3.26) fit in with this summation. In this case, however, $f_n(\cdot)$ is learned by the SSNN (in our other approaches, we explicitly modeled the relationship between x_k and p_k using a sigmoid). Similarly, if a continuous-valued variable s_k is present in y_k, there is the summation

$$\sum \frac{1}{2} \log \left[2\pi f_{\sigma_s^2}(x_k) \right] + \frac{\left[s_k - f_{\mu_s}(x_k) \right]^2}{2 f_{\sigma_s^2}(x_k)}, \tag{9.46}$$

where $f_{\mu_s}(\cdot)$ and $f_{\sigma_s^2}(\cdot)$ represent mean and variance functions learned by the SSNN. Again, recall that we had a very similar term at the parameter estimation step for a continuous variable s_k.

One of the primary advantages of the neural network approach in [116] is that we no longer need to derive all the EM algorithm equations when new observations are added. This is a notable drawback with the traditional EM approach. Moreover, we can also modify the objective function to

$$(1-\rho) Q_{ML} + \rho \sum (x_k - l_k)^2, \tag{9.47}$$

where l_k is an external influence and $0 \leq \rho \leq 1$. This provides the option to perform state estimation while permitting an external influence (e.g., domain knowledge or subject-provided labels) to affect x_k.

9.5 Additional MATLAB Code Examples

In this section we briefly describe the two state-space models in [118] and [30] for which the MATLAB code examples are provided. The equation derivations for these two models require no significant new knowledge. The first of these incorporates one binary observation from skin conductance and one EKG spiking-type observation. The second incorporates one binary observation and two continuous

observations. It is almost identical to the model with the same observations described in an earlier chapter but has a circadian rhythm term as I_k. The derivation of the state and parameter estimation equations is similar to what we have seen before.

9.5.1 State-Space Model with One Binary and One Spiking-Type Observation

The MATLAB code example for the state-space model with one binary and one spiking-type observation is provided in the "one_bin_one_spk" folder. The model is described in [118] and attempts to estimate sympathetic arousal from binary-valued SCRs and EKG R-peaks (the RR-intervals are modeled using an HDIG-based CIF). The results are shown in Fig. 9.1. The data come from the study described in [119] where subjects had to perform office work-like tasks under different conditions. In the first condition, the subjects were permitted to take as much time as they liked. The other two conditions involved e-mail interruptions and time constraints. Based on the results reported in [118], it appeared that task uncertainty (i.e., how new the task is) seemed to have generated the highest sympathetic arousal responses for the subject considered.

9.5.2 State-Space Model with One Binary and Two Continuous Observations with a Circadian Input in the State Equation

Cortisol is known to exhibit circadian variation [120, 121]. Typically, cortisol concentrations in the blood begin to rise early morning during late stages of sleep. Peak values are reached shortly after awakening. Later in the day, cortisol levels tend to drop toward bedtime and usually reach their lowest values in the middle of the night [122, 123]. In [30], a circadian I_k term was assumed to drive x_k so that it evolved with time following

$$x_k = \rho x_{k-1} + I_k + \varepsilon_k, \tag{9.48}$$

where

$$I_k = \sum_{i=1}^{2} a_i \sin\left(\frac{2\pi ik}{1440}\right) + b_i \cos\left(\frac{2\pi ik}{1440}\right). \tag{9.49}$$

The model also considered the upper and lower envelopes of the blood cortisol concentrations as the two continuous variables r_k and s_k. The pulsatile secretions

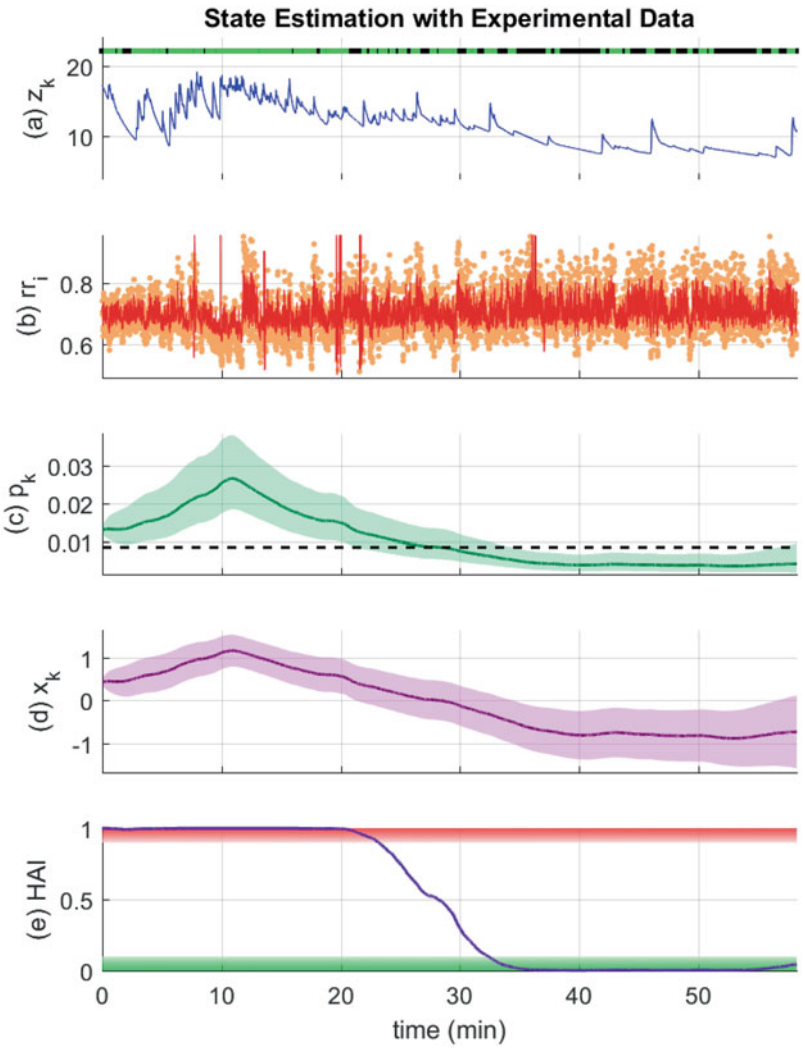

Fig. 9.1 State estimation based on observing one binary and one spiking-type variable. The subpanels, respectively, depict (**a**) the skin conductance signal z_k (the green and black dots on top depict the presence or the absence of SCRs, respectively), (**b**) the RR-interval sequence (orange) and the fit to the HDIG mean (red), (**c**) the probability of SCR occurrence p_k and its 95% confidence limits, (**d**) the arousal state x_k and its 95% confidence limits, and (**e**) the HAI (the regions above 90% and below 10% are shaded in red and green, respectively). © 2019 IEEE. Reprinted, with permission, from [118]

formed the binary variable n_k. The inclusion of each continuous variable necessitates the determination of three model parameters (two governing the linear fit and the third being the sensor noise variance). In addition, the state-space model in [30] also estimated β_0 and β_1 in p_k. There are also six more parameters in the

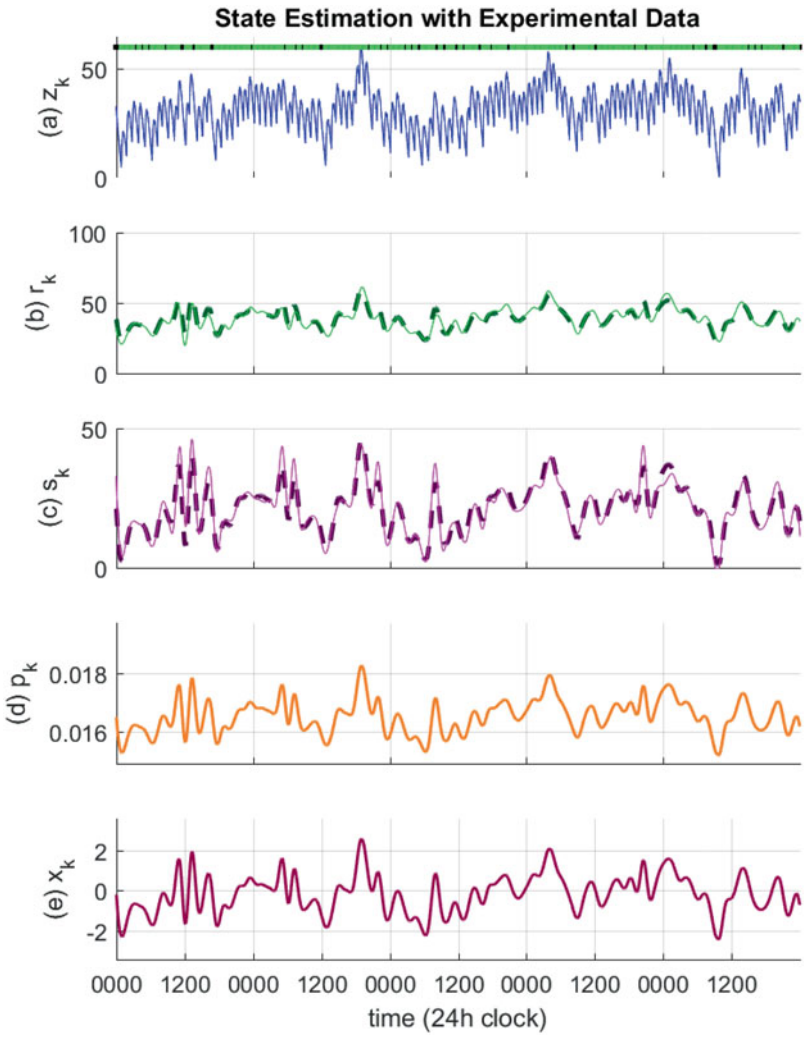

Fig. 9.2 State estimation based on observing one binary and two continuous variables with a circadian input in the state equation. The sub-panels, respectively, depict (**a**) the cortisol profile (the green and black dots on top denote the presence or the absence of pulsatile secretions respectively), (**b**) the first cortisol concentration envelope r_k (green solid) and its estimate (dashed), (**c**) the second cortisol concentration envelope s_k (mauve solid) and its estimate (dashed), (**d**) the probability of pulse occurrence p_k, and (**e**) the energy state x_k. © 2019 IEEE. Reprinted, with permission, from [30]

state equation: ρ, a_1, a_2, b_1, b_2, and σ_ε^2. To ease computational complexity, the EM algorithm in [30] treated the four parameters related to the circadian rhythm (a_1, a_2, b_1, and b_2) somewhat differently. Thus, while all the parameters were updated at the parameter estimation step, a_1, a_2, b_1, and b_2 were excluded from the convergence criteria. The results are shown in Fig. 9.2. Here, the data were simulated

for a hypothetical patient suffering from a type of hypercortisolism (Cushing's disease) based on the parameters in [124]. Cushing's disease involves excess cortisol secretion into the bloodstream and may be caused by tumors or prolonged drug use [125]. Symptoms of Cushing's disease involve a range of physical and psychological symptoms including insomnia and fatigue [126–128]. The resulting cortisol-related energy state estimates do not have the usual circadian-like patterns seen for a healthy subject. This may partially account for why Cushing's patients experience daytime bouts of fatigue and nighttime sleeping difficulties.

Chapter 10
MATLAB Code Examples

10.1 State-space Model with One Binary Observation

10.1.1 Simulated Data Example

```
load('data_one_bin.mat');

K = length(n);

M = 2e4;
ve = zeros(1, M);   % process noise variance

x_pred = zeros(1, K);
v_pred = zeros(1, K);

x_updt = zeros(1, K);
v_updt = zeros(1, K);

x_smth = zeros(1, K);
v_smth = zeros(1, K);

p_updt = zeros(1, K);

base_prob = sum(n) / length(n);
tol = 1e-6; % convergence criteria

A = zeros(1, K);
W = zeros(1, K);
CW = zeros(1, K);
C = zeros(1, K);

ve(1) = 0.005;
x_smth(1) = 0;
b0 = log(base_prob / (1 - base_prob));
```


D. S. Wickramasuriya, R. T. Faghih, *Bayesian Filter Design for Computational Medicine*, https://doi.org/10.1007/978-3-031-47104-9_10

```
for m = 1:M

    for k = 1:K

        if (k == 1) % boundary condition
            x_pred(k) = x_smth(1);
            v_pred(k) = ve(m) + ve(m);
        else
            x_pred(k) = x_updt(k - 1);
            v_pred(k) = v_updt(k - 1) + ve(m);
        end

        x_updt(k) = get_state_update(x_pred(k), v_pred(k), b0, n(
    k));

        p_updt(k) = 1 / (1 + exp((-1) * (b0 + x_updt(k))));
        v_updt(k) = 1 / ((1 / v_pred(k)) + p_updt(k) * (1 -
    p_updt(k)));
    end

    x_smth(K) = x_updt(K);
    v_smth(K) = v_updt(K);
    W(K) = v_smth(K) + (x_smth(K) ^ 2);

    A(1:(end - 1)) = v_updt(1:(end - 1)) ./ v_pred(2:end);

    for k = (K - 1):(-1):1
       x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
     1));
       v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
    v_pred(k + 1));

       CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
       W(k) = v_smth(k) + (x_smth(k) ^ 2);
    end

    if (m < M)

        ve(m + 1) = (sum(W(2:end)) + sum(W(1:(end - 1))) - 2 *
    sum(CW)) / K;
        mean_dev = mean(abs(ve(m + 1) - ve(m)));

        if mean_dev < tol
            fprintf('m = %d\nx0 = %.18f\nve = %.18f\n\n', m,
    x_smth(1), ve(m));
            fprintf('Converged at m = %d\n\n', m);
            break;
        else
            fprintf('m = %d\nx0 = %.18f\nve = %.18f\n\n', m,
    x_smth(1), ve(m + 1));

            x_pred = zeros(1, K);
            v_pred = zeros(1, K);
```

```
            x_updt = zeros(1, K);
            v_updt = zeros(1, K);

            x_smth(2:end) = zeros(1, K - 1);    % x_smth(1)
    needed for next iteration
            v_smth = zeros(1, K);

            p_updt = zeros(1, K);

            A = zeros(1, K);
            W = zeros(1, K);
            CW = zeros(1, K);
            C = zeros(1, K);
        end
    end
end

p_smth = 1 ./ (1 + exp((-1) * (b0 + x_smth)));

figure;
subplot(411);
stem(n, 'fill', 'color', [0 0.75 0]);
ylim([0 1.25]);
ylabel('(a) n_{k}');
grid; title('Estimation with Simulated Data');

subplot(412);
hold on;
plot(p, 'b');
plot(p_smth, 'r-.', 'linewidth', 1.25);
ylabel('(b) p_{k}');
grid;

subplot(413);
hold on;
plot(x, 'b');
plot(x_smth, 'r-.', 'linewidth', 1.25);
ylabel('(c) x_{k}'); xlabel('time index');
grid;

subplot(414);
qqplot(x - x_smth);
title('QQ Plot - State Estimate', 'FontWeight', 'Normal');
ylabel('(d) input quantiles');
xlabel('standard normal quantiles');
grid;

function [y] = get_state_update(x_pred, v_pred, b0, n)

    M = 50;    % maximum iterations

    it = zeros(1, M);
    func = zeros(1, M);
```

```
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)
       func(i) = it(i) - x_pred - v_pred * (n - exp(b0 + it(i)) /
     (1 + exp(b0 + it(i))));
       df(i) = 1 + v_pred * exp(b0 + it(i)) / ((1 + exp(b0 + it(i
    ))) ^ 2);
       it(i + 1)  = it(i) - func(i) / df(i);

       if abs(it(i + 1) - it(i)) < 1e-14
           y = it(i + 1);
          return
       end
    end

    error('Newton-Raphson failed to converge.');

end
```

10.1.2 Experimental Data Example

```
load('expm_data_one_bin.mat');

K = length(u);
n = zeros(1, K);

pt = find(u > 0);
n(pt) = 1;

M = 2e4;
ve = zeros(1, M);   % process noise variance

x_pred = zeros(1, K);
v_pred = zeros(1, K);

x_updt = zeros(1, K);
v_updt = zeros(1, K);

x_smth = zeros(1, K);
v_smth = zeros(1, K);

p_updt = zeros(1, K);

base_prob = sum(n) / length(n);
tol = 1e-8; % convergence criteria

A = zeros(1, K);
W = zeros(1, K);
CW = zeros(1, K);
C = zeros(1, K);
```

```
ve(1) = 0.005;
x_smth(1) = 0;
b0 = log(base_prob / (1 - base_prob));

for m = 1:M

    for k = 1:K

        if (k == 1) % boundary condition
            x_pred(k) = x_smth(1);
            v_pred(k) = ve(m) + ve(m);
        else
            x_pred(k) = x_updt(k - 1);
            v_pred(k) = v_updt(k - 1) + ve(m);
        end

        x_updt(k) = get_state_update(x_pred(k), v_pred(k), b0, n(
    k));

        p_updt(k) = 1 / (1 + exp((-1) * (b0 + x_updt(k))));
        v_updt(k) = 1 / ((1 / v_pred(k)) + p_updt(k) * (1 -
    p_updt(k)));
    end

    x_smth(K) = x_updt(K);
    v_smth(K) = v_updt(K);
    W(K) = v_smth(K) + (x_smth(K) ^ 2);

    A(1:(end - 1)) = v_updt(1:(end - 1)) ./ v_pred(2:end);
    x0_prev = x_smth(1);

    for k = (K - 1):(-1):1
       x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
     1));
       v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
    v_pred(k + 1));

       CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
       W(k) = v_smth(k) + (x_smth(k) ^ 2);
    end

    if (m < M)

        ve(m + 1) = (sum(W(2:end)) + sum(W(1:(end - 1))) - 2 *
    sum(CW) + 0.5 * W(1)) / (K + 1);
        x0 = x_smth(1) / 2;

        if (abs(ve(m + 1) - ve(m)) < tol) && (abs(x0 - x0_prev) <
     tol)
            fprintf('m = %d\nx0 = %.18f\nve = %.18f\n\n', m,
    x_smth(1), ve(m));
            fprintf('Converged at m = %d\n\n', m);
            break;
```

```
        else
            fprintf('m = %d\nx0 = %.18f\nve = %.18f\n\n', m,
    x_smth(1), ve(m + 1));

            x_pred = zeros(1, K);
            v_pred = zeros(1, K);

            x_updt = zeros(1, K);
            v_updt = zeros(1, K);

            x_smth(2:end) = zeros(1, K - 1);    % x_smth(1)
    needed for next iteration
            x_smth(1) = x0;
            v_smth = zeros(1, K);

            p_updt = zeros(1, K);

            A = zeros(1, K);
            W = zeros(1, K);
            CW = zeros(1, K);
            C = zeros(1, K);
        end
    end
end

p_smth = 1 ./ (1 + exp((-1) * (b0 + x_smth)));

lcl_x = norminv(0.025, x_smth, sqrt(v_smth));
ucl_x = norminv(0.975, x_smth, sqrt(v_smth));

certainty = 1 - normcdf(prctile(x_smth, 50) * ones(1, length(
    x_smth)), x_smth, sqrt(v_smth));

lcl_p = zeros(1, K);
ucl_p = zeros(1, K);

disp('Calculating the pk confidence limits... (this can take time
     due to the resolution)');
for k = 1:K
    [lcl_p(k), ucl_p(k)] = get_pk_conf_lims(v_smth(k), b0, x_smth
    (k));
end
disp('Finished calculating the pk confidence limits.');

fs = 4;
t = (0:(K - 1)) / fs;
tr = ((K - 1):(-1):0) / fs;

u_plot = NaN * ones(1, K);
u_plot(pt) = u(pt);

subplot(511);
hold on;
plot(ty, y, 'k', 'linewidth', 1.25);
```

```
ylabel({'(a) skin cond.', '(\mu S)'});
set(gca,'xticklabel', []); ylim([0 3]);
title('State Estimation with Experimental Data'); xlim([0 ty(end)
    ]);
grid;
yl = ylim;

patch([xp(1), xp(2), xp(2), xp(1)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'r', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(2), xp(3), xp(3), xp(2)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'g', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(3), xp(4), xp(4), xp(3)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], [1 0.647059 0], 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(4), xp(5), xp(5), xp(4)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(5), xp(6), xp(6), xp(5)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'y', 'FaceAlpha', 0.2, 'EdgeColor', 'none');

subplot(512);
stem(t, u_plot, 'fill', 'k', 'markersize', 3);
ylabel('(b) n_{k}, r_{k}'); grid; xlim([0 t(end)]); ylim([0 15]);
yl = ylim; set(gca,'xticklabel', []);

patch([xp(1), xp(2), xp(2), xp(1)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'r', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(2), xp(3), xp(3), xp(2)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'g', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(3), xp(4), xp(4), xp(3)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], [1 0.647059 0], 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(4), xp(5), xp(5), xp(4)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(5), xp(6), xp(6), xp(5)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'y', 'FaceAlpha', 0.2, 'EdgeColor', 'none');

subplot(513);
hold on;
plot(t, x_smth, 'b', 'linewidth', 1.25);
fill([t, tr], [lcl_x fliplr(ucl_x)], 'c', 'EdgeColor', 'none', '
    FaceAlpha', 0.5);
ylabel('(c) state (x_{k})'); ylim([-10 5]);
set(gca,'xticklabel', []); xlim([0 t(end)]);
grid; yl = ylim;

patch([xp(1), xp(2), xp(2), xp(1)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'r', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(2), xp(3), xp(3), xp(2)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'g', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(3), xp(4), xp(4), xp(3)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], [1 0.647059 0], 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(4), xp(5), xp(5), xp(4)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(5), xp(6), xp(6), xp(5)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'y', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
```

```
subplot(514);
hold on;
plot(t, p_smth, 'r', 'linewidth', 1.5);
fill([t, tr], [lcl_p fliplr(ucl_p)], [1, 0, (127 / 255)], '
    EdgeColor', 'none', 'FaceAlpha', 0.3);
ylim([0 0.15]);
ylabel('(d) probability (p_{k})');
set(gca,'xticklabel', []); xlim([0 t(end)]);
grid; yl = ylim;

patch([xp(1), xp(2), xp(2), xp(1)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'r', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(2), xp(3), xp(3), xp(2)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'g', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(3), xp(4), xp(4), xp(3)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], [1 0.647059 0], 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(4), xp(5), xp(5), xp(4)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(5), xp(6), xp(6), xp(5)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'y', 'FaceAlpha', 0.2, 'EdgeColor', 'none');

subplot(515);
hold on;
v1 = [0 0.9; t(end) 0.9; t(end) 1; 0 1];
c1 = [1 (220 / 255) (220 / 255); 1 (220 / 255) (220 / 255); 1 0
    0; 1 0 0];
faces1 = [1 2 3 4];

patch('Faces', faces1, 'Vertices', v1, 'FaceVertexCData', c1, '
    FaceColor', 'interp', ...
    'EdgeColor', 'none', 'FaceAlpha', 0.7);

v2 = [0 0; t(end) 0; t(end) 0.1; 0 0.1];
c2 = [0 0.8 0; 0 0.8 0; (204 / 255) 1 (204 / 255); (204 / 255) 1
    (204 / 255)];
faces2 = [1 2 3 4];

patch('Faces', faces2, 'Vertices', v2, 'FaceVertexCData', c2, '
    FaceColor', 'interp', ...
    'EdgeColor', 'none', 'FaceAlpha', 0.7);

plot(t, certainty, 'color', [(138 / 255) (43 / 255) (226 / 255)],
     'linewidth', 1.5); grid;
ylabel('(d) HAI'); xlabel('time (s)'); xlim([0 t(end)]);

function [y] = get_state_update(x_pred, v_pred, b0, n)

    M = 50;    % maximum iterations

    it = zeros(1, M);
    func = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;
```

```
    for i = 1:(M - 1)
       func(i) = it(i) - x_pred - v_pred * (n - exp(b0 + it(i)) /
     (1 + exp(b0 + it(i))));
       df(i) = 1 + v_pred * exp(b0 + it(i)) / ((1 + exp(b0 + it(i
    ))) ^ 2);
       it(i + 1)  = it(i) - func(i) / df(i);

       if abs(it(i + 1) - it(i)) < 1e-14
           y = it(i + 1);
          return
       end
    end

    error('Newton-Raphson failed to converge.');

end

function [lcl, ucl] = get_pk_conf_lims(v, b0, x)

    p = (1e-6:1e-6:1);

    fp = cumtrapz(p, 1 ./ (sqrt(2 * pi * v) * p .* (1 - p)) .*
    ...
        exp(((-1) / (2 * v))* (log(p ./ ((1 - p) * exp(b0))) - x)
     .^ 2));

    n = find(fp <= 0.975);
    m = find(fp < 0.025);

    ucl = p(n(end));
    lcl = p(m(end));
end
```

10.2 State-space Model with One Binary and One Continuous Observation

10.2.1 Simulated Data Example

```
load('data_one_bin_one_cont.mat');

K = length(n);

pt = find(n > 0);

M = 5e4;
ve = zeros(1, M);   % process noise variance
r0 = zeros(1, M);   % linear model coefficients (continuous
    variable)
```

```
r1 = zeros(1, M);   % linear model coefficients (continuous
    variable)
vr = zeros(1, M);   % sensor noise variance (continuous variable)

x_pred = zeros(1, K);
v_pred = zeros(1, K);

x_updt = zeros(1, K);
v_updt = zeros(1, K);

x_smth = zeros(1, K);
v_smth = zeros(1, K);

p_updt = zeros(1, K);

base_prob = sum(n) / length(n);
tol = 1e-8; % convergence criteria

A = zeros(1, K);
W = zeros(1, K);
CW = zeros(1, K);
C = zeros(1, K);

ve(1) = 0.005;
x_smth(1) = 0;
r0(1) = 0.1;
r1(1) = r(1);
vr(1) = 0.002;
b0 = log(base_prob / (1 - base_prob));

for m = 1:M

    for k = 1:K

        if (k == 1)
            x_pred(k) = x_smth(1);
            v_pred(k) = ve(m) + ve(m);
        else
            x_pred(k) = x_updt(k - 1);
            v_pred(k) = v_updt(k - 1) + ve(m);
        end
        x_updt(k) = get_posterior_mode(x_pred(k), v_pred(k), r(k)
    , r0(m), r1(m), vr(m), b0, n(k));
        p_updt(k) = 1 / (1 + exp((-1) * (b0 + x_updt(k))));
        v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(m))
    + p_updt(k) * (1 - p_updt(k)));
    end

    x_smth(K) = x_updt(K);
    v_smth(K) = v_updt(K);
    W(K) = v_smth(K) + (x_smth(K) ^ 2);

    A(1:(end - 1)) = v_updt(1:(end - 1)) ./ v_pred(2:end);
```

```
    for k = (K - 1):(-1):1
       x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
     1));
       v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
    v_pred(k + 1));

       CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
       W(k) = v_smth(k) + (x_smth(k) ^ 2);
    end

    if (m < M)

        R = get_linear_parameters(x_smth, W, r);

        r0(m + 1) = R(1, 1);
        r1(m + 1) = R(2, 1);
        vr(m + 1) = get_maximum_variance(r, r0(m + 1), r1(m + 1),
     W, x_smth);

        ve(m + 1) = (sum(W(2:end)) + sum(W(1:(end - 1))) - 2 *
    sum(CW)) / K;

        mean_dev = mean(abs([ve(m + 1) r0(m + 1) r1(m + 1) vr(m +
     1)] - [ve(m) r0(m) r1(m) vr(m)]));

        if mean_dev < tol
            fprintf('m = %d\nx0 = %.18f\n\nr0 = %.18f\nr1 = %.18f
    \nvr = %.18f\nve = %.18f\n\n', m, x_smth(1), r0(m), r1(m), vr
    (m), ve(m));
            fprintf('Converged at m = %d\n\n', m);
            break;
        else
            fprintf('m = %d\nx0 = %.18f\n\nr0 = %.18f\nr1 = %.18f
    \nvr = %.18f\nve = %.18f\n\n', m, x_smth(1), r0(m + 1), r1(m
    + 1), vr(m + 1), ve(m + 1));

            x_pred = zeros(1, K);
            v_pred = zeros(1, K);

            x_updt = zeros(1, K);
            v_updt = zeros(1, K);

            x_smth(2:end) = zeros(1, K - 1);    % x_smth(1)
    needed for next iteration
            v_smth = zeros(1, K);

            p_updt = zeros(1, K);

            A = zeros(1, K);
            W = zeros(1, K);
            CW = zeros(1, K);
            C = zeros(1, K);
        end
    end
```

```
end

p_updt = 1 ./ (1 + exp((-1) * (b0 + x_updt)));
p_smth = 1 ./ (1 + exp((-1) * (b0 + x_smth)));
r_smth = r0(m) + r1(m) * x_smth;

figure;
subplot(511);
stem(n, 'fill', 'color', [0 0.75 0]);
ylim([0 1.25]);
ylabel('(a) n_{k}');
grid; title('Estimation with Simulated Data');

subplot(512);
hold on;
plot(r, 'b');
plot(r_smth, 'r-.', 'linewidth', 1.5);
ylabel('(b) r_{k}');
grid;

subplot(513);
hold on;
plot(p, 'b');
plot(p_smth, 'r-.', 'linewidth', 1.5);
ylabel('(c) p_{k}');
grid;

subplot(514);
hold on;
plot(x, 'b');
plot(x_smth, 'r-.', 'linewidth', 1.5);
ylabel('(d) x_{k}');
xlabel('time index');
grid;

subplot(515);
qqplot(x - x_smth);
title('QQ Plot - State Estimate', 'FontWeight', 'Normal');
ylabel('(e) input quantiles');
xlabel('standard normal quantiles');
grid;

function [y] = get_posterior_mode(x_pred, v_pred, z, r0, r1, vr,
    b0, n)

    M = 100;    % maximum iterations

    it = zeros(1, M);
    f = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)
```

```
        C = v_pred / ((r1 ^ 2) * v_pred + vr);
        f(i) = it(i) - x_pred - C * (r1 * (z - r0 - r1 * x_pred)
    + vr * (n - (1 / (1 + exp((-1) * (b0 + it(i)))))));
        df(i) = 1 + C * vr * exp(b0 + it(i)) / ((1 + exp(b0 + it(
    i))) ^ 2);

        it(i + 1)  = it(i) - f(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
            y = it(i + 1);
            return
        end
    end

    error('Newton-Raphson failed to converge.');

end

function y = get_maximum_variance(z, r0, r1, W, x_smth)

    K = length(x_smth);

    y = (z * z' + K * (r0 ^ 2) + (r1 ^ 2) * sum(W) ...
                - 2 * r0 * sum(z) - 2 * r1 * dot(x_smth, z) + 2 *
     r0 * r1 * sum(x_smth)) / K;
end

function y = get_linear_parameters(x_smth, W, z)

    K = length(x_smth);

    y = [K sum(x_smth); sum(x_smth) sum(W)] \ [sum(z); sum(z .*
    x_smth)];

end
```

10.2.2 Experimental Data Example

```
load('expm_data_one_bin_one_cont.mat');

K = length(n);

pt = find(n > 0);

M = 5e4;
ve = zeros(1, M);   % process noise variance
r0 = zeros(1, M);   % linear model coefficients (continuous
    variable)
r1 = zeros(1, M);   % linear model coefficients (continuous
    variable)
vr = zeros(1, M);   % sensor noise variance (continuous variable)
```

```
x_pred = zeros(1, K);
v_pred = zeros(1, K);

x_updt = zeros(1, K);
v_updt = zeros(1, K);

x_smth = zeros(1, K);
v_smth = zeros(1, K);

p_updt = zeros(1, K);

base_prob = sum(n) / length(n);
tol = 1e-8; % convergence criteria

A = zeros(1, K);
W = zeros(1, K);
CW = zeros(1, K);
C = zeros(1, K);

ve(1) = 0.005;
x_smth(1) = 0;
r0(1) = 0.1;
r1(1) = r(1);
vr(1) = 0.002;
b0 = log(base_prob / (1 - base_prob));

for m = 1:M

    for k = 1:K

        if (k == 1)
            x_pred(k) = x_smth(1);
            v_pred(k) = ve(m) + ve(m);
        else
            x_pred(k) = x_updt(k - 1);
            v_pred(k) = v_updt(k - 1) + ve(m);
        end
        x_updt(k) = get_posterior_mode(x_pred(k), v_pred(k), r(k)
    , r0(m), r1(m), vr(m), b0, n(k));
        p_updt(k) = 1 / (1 + exp((-1) * (b0 + x_updt(k))));
        v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(m))
    + p_updt(k) * (1 - p_updt(k)));
    end

    x_smth(K) = x_updt(K);
    v_smth(K) = v_updt(K);
    W(K) = v_smth(K) + (x_smth(K) ^ 2);

    A(1:(end - 1)) = v_updt(1:(end - 1)) ./ v_pred(2:end);

    for k = (K - 1):(-1):1
       x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
     1));
```

```
        v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
    v_pred(k + 1));

        CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
        W(k) = v_smth(k) + (x_smth(k) ^ 2);
    end

    if (m < M)

        R = get_linear_parameters(x_smth, W, r);

        r0(m + 1) = R(1, 1);
        r1(m + 1) = R(2, 1);
        vr(m + 1) = get_maximum_variance(r, r0(m + 1), r1(m + 1),
     W, x_smth);

        ve(m + 1) = (sum(W(2:end)) + sum(W(1:(end - 1))) - 2 *
    sum(CW)) / K;

        mean_dev = mean(abs([ve(m + 1) r0(m + 1) r1(m + 1) vr(m +
     1)] - [ve(m) r0(m) r1(m) vr(m)]));

        if mean_dev < tol
            fprintf('m = %d\nx0 = %.18f\n\nr0 = %.18f\nr1 = %.18f
    \nvr = %.18f\nve = %.18f\n\n', m, x_smth(1), r0(m), r1(m), vr
    (m), ve(m));
            fprintf('Converged at m = %d\n\n', m);
            break;
        else
            fprintf('m = %d\nx0 = %.18f\n\nr0 = %.18f\nr1 = %.18f
    \nvr = %.18f\nve = %.18f\n\n', m, x_smth(1), r0(m + 1), r1(m
    + 1), vr(m + 1), ve(m + 1));

            x_pred = zeros(1, K);
            v_pred = zeros(1, K);

            x_updt = zeros(1, K);
            v_updt = zeros(1, K);

            x_smth(2:end) = zeros(1, K - 1);    % x_smth(1)
    needed for next iteration
            v_smth = zeros(1, K);

            p_updt = zeros(1, K);

            A = zeros(1, K);
            W = zeros(1, K);
            CW = zeros(1, K);
            C = zeros(1, K);
        end
    end
end

p_updt = 1 ./ (1 + exp((-1) * (b0 + x_updt)));
```

```
p_smth = 1 ./ (1 + exp((-1) * (b0 + x_smth)));
r_smth = r0(m) + r1(m) * x_smth;
n_plot = NaN * ones(1, K);
n_plot(n > 0) = 1;

figure;

subplot(511);
hold on;
plot(t, y, 'k', 'linewidth', 1.5);
patch(xp, yp, [192, 192, 192] / 255, 'EdgeColor', 'none', '
    FaceAlpha', 0.3);
grid;
ylabel('(a) z_{k}'); title('State Estimation with Experimental
    Data');
set(gca,'xticklabel', []); ylim([(min(y) - 1e2) (max(y) + 1e2)]);

subplot(512);
stem(t, n_plot, 'fill', 'color', [0 0.75 0]);
patch(xp, yp * 1.25, [192, 192, 192] / 255, 'EdgeColor', 'none',
    'FaceAlpha', 0.3);
ylim([0 1.25]);
ylabel('(b) n_{k}');
grid; set(gca,'xticklabel', []);

subplot(513);
hold on;
plot(t, r, 'b');
plot(t, r_smth, 'r', 'linewidth', 1.5);
patch(xp, yp, [192, 192, 192] / 255, 'EdgeColor', 'none', '
    FaceAlpha', 0.3);
ylabel('(c) r_{k}');
grid; set(gca,'xticklabel', []);

subplot(514);
hold on;
plot(t, p_smth, 'color', [(204 / 255), 0, (102 / 255)], '
    linewidth', 1.25);
patch(xp, yp * 0.5, [192, 192, 192] / 255, 'EdgeColor', 'none', '
    FaceAlpha', 0.3);
ylabel('(d) p_{k}');
grid; set(gca,'xticklabel', []);

subplot(515);
hold on;
plot(t, x_smth, 'color', [(153 / 255), 0, (153 / 255)], '
    linewidth', 1.25);
yl = ylim;
ypx = yp;
ypx(yp == 0) = yl(1);
ypx(yp == 1) = yl(2);
patch(xp, ypx, [192, 192, 192] / 255, 'EdgeColor', 'none', '
    FaceAlpha', 0.3);
ylabel('(e) x_{k}');
```

```
xlabel('time (min)');
grid; ylim([(min(x_smth) - 2) inf]);

function [y] = get_posterior_mode(x_pred, v_pred, z, r0, r1, vr,
    b0, n)

    M = 100;    % maximum iterations

    it = zeros(1, M);
    f = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)
        C = v_pred / ((r1 ^ 2) * v_pred + vr);
        f(i) = it(i) - x_pred - C * (r1 * (z - r0 - r1 * x_pred)
    + vr * (n - (1 / (1 + exp((-1) * (b0 + it(i)))))));
        df(i) = 1 + C * vr * exp(b0 + it(i)) / ((1 + exp(b0 + it(
    i))) ^ 2);

        it(i + 1)  = it(i) - f(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
            y = it(i + 1);
            return
        end
    end

    error('Newton-Raphson failed to converge.');

end

function y = get_maximum_variance(z, r0, r1, W, x_smth)

    K = length(x_smth);

    y = (z * z' + K * (r0 ^ 2) + (r1 ^ 2) * sum(W) ...
                - 2 * r0 * sum(z) - 2 * r1 * dot(x_smth, z) + 2 *
     r0 * r1 * sum(x_smth)) / K;
end

function y = get_linear_parameters(x_smth, W, z)

    K = length(x_smth);

    y = [K sum(x_smth); sum(x_smth) sum(W)] \ [sum(z); sum(z .*
    x_smth)];

end
```

10.3 State-space Model with One Binary and Two Continuous Observations

10.3.1 Simulated Data Example (αI_k Excluded)

```
load('data_one_bin_two_cont_no_extern_stim.mat');

base_prob = sum(n) / length(n);

%% parameters

M = 1e6;    % maximum iterations
m = 1;
tol = 1e-8; % convergence criteria

b0 = zeros(1, M);   % binary GLM model
b1 = zeros(1, M);

r0 = zeros(1, M);   % continuous model
r1 = zeros(1, M);
vr = zeros(1, M);   % continuous model noise variance (1)

s0 = zeros(1, M);   % continuous model
s1 = zeros(1, M);
vs = zeros(1, M);   % continuous model noise variance (2)

ve = zeros(1, M);   % process noise variance
rho = zeros(1, M);  % random walk forgetting factor

K = length(n);

x_pred = zeros(1, K);
v_pred = zeros(1, K);

x_updt = zeros(1, K);
v_updt = zeros(1, K);

x_smth = zeros(1, K);
v_smth = zeros(1, K);

p_updt = zeros(1, K);

A = zeros(1, K);
W = zeros(1, K);
CW = zeros(1, K);
C = zeros(1, K);

%% initial guesses

b0(1) = log(base_prob / (1 - base_prob));
b1(1) = 1;
```

```
r0(1) = r(1); % guess it's the first value of r
r1(1)  = 0.5;
s0(1) = s(1);
s1(1) = 1;
vr(1) = 0.05;
vs(1) = 0.05;
ve(1) = 0.05;
rho(1) = 1;

%% main function

for m = 1:M

    for k = 1:K

        if (k == 1)
            x_pred(k) = x_smth(1);
            v_pred(k) = ve(m) + ve(m);
        else
            x_pred(k) = rho(m) * x_updt(k - 1);
            v_pred(k) = (rho(m) ^ 2) * v_updt(k - 1) + ve(m);
        end

        C(k) = v_pred(k) / (vr(m) * vs(m) + v_pred(k) * ((r1(m) ^
     2) * vs(m) + (s1(m) ^ 2) * vr(m)));
        x_updt(k) = get_posterior_mode(x_pred(k), C(k), r(k), r0(
    m), r1(m), b0(m), b1(m), vr(m), n(k), s(k), s0(m), s1(m), vs(
    m));

        p_updt(k) = 1 / (1 + exp((-1) * (b0(m) + b1(m) * x_updt(k
    ))));
        v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(m))
    + ((s1(m) ^ 2) / vs(m)) + (b1(m) ^ 2) * p_updt(k) * (1 -
    p_updt(k)));
    end

    x_smth(K) = x_updt(K);
    v_smth(K) = v_updt(K);
    W(K) = v_smth(K) + (x_smth(K) ^ 2);

    A(1:(end - 1)) = rho(m) * v_updt(1:(end - 1)) ./ v_pred(2:end
    );

    for k = (K - 1):(-1):1
       x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
     1));
       v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
    v_pred(k + 1));

       CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
       W(k) = v_smth(k) + (x_smth(k) ^ 2);
    end

    prev = [r0(m) r1(m) ve(m) vr(m) rho(m) s0(m) s1(m) vs(m)];
```

```
R = get_linear_parameters(x_smth, W, r, K);
S = get_linear_parameters(x_smth, W, s, K);

b0(m + 1) = log(base_prob / (1 - base_prob));
b1(m + 1) = 1;

rho(m + 1) = sum(CW) / sum(W(1:end - 1));

ve(m + 1) = (sum(W(2:end)) + (rho(m +  1) ^ 2) * sum(W(1:(end
 - 1))) - 2 * rho(m + 1) * sum(CW)) / K;

r0(m + 1) = R(1, 1);
r1(m + 1) = R(2, 1);

s0(m + 1) = S(1, 1);
s1(m + 1) = S(2, 1);

vr(m + 1) = get_maximum_variance(r, r0(m + 1), r1(m + 1), W,
x_smth, K);
vs(m + 1) = get_maximum_variance(s, s0(m + 1), s1(m + 1), W,
x_smth, K);

next = [r0(m + 1) r1(m + 1) ve(m + 1) vr(m + 1) rho(m + 1) s0
(m + 1) s1(m + 1) vs(m +1)];

mean_dev = mean(abs(next - prev));

if mean_dev < tol
    fprintf('Converged at m = %d\n\n', m);
    break;
else
    fprintf('m = %d\nr0 = %.18f\nr1 = %.18f\nvr = %.18f\n\ns0
 = %.18f\ns1 = %.18f\nvs = %.18f\n\nve = %.18f\nrho = %.18f\n
\n', ...
    m + 1, r0(m + 1), r1(m + 1), vr(m + 1), s0(m + 1), s1(m +
 1), vs(m + 1), ve(m + 1), rho(m + 1));

    x_pred = zeros(1, K);
    v_pred = zeros(1, K);

    x_updt = zeros(1, K);
    v_updt = zeros(1, K);

    x_smth(2:end) = zeros(1, K - 1);    % x_smth(1) needed
for next iteration
    v_smth = zeros(1, K);

    p_updt = zeros(1, K);

    A = zeros(1, K);
    W = zeros(1, K);
    CW = zeros(1, K);
    C = zeros(1, K);
```

```
    end
end

%% calculate confidence limits

p_smth = 1 ./ (1 + exp((-1) * (b0(m) + b1(m) * x_smth)));  % mode
    , lower and upper confidence limits for binary distribution
r_smth = r0(m) + r1(m) * x_smth;
s_smth = s0(m) + s1(m) * x_smth;

%% plot graphs

figure;

subplot(511);
hold on;
plot(p, 'b');
plot(p_smth, 'r-.', 'linewidth', 1); grid;
plot(find(n == 0) - 1, 1.4 * max(p) * ones(length(find(n == 0))),
    'ks', 'MarkerFaceColor', 'k', 'MarkerSize', 4);
plot(find(n == 1) - 1, 1.4 * max(p) * ones(length(find(n == 1))),
    'gs', 'MarkerFaceColor', 'g', 'MarkerSize', 4);
ylabel('(a) p_{k}'); ylim([0 0.18]);
title('State Estimation with Simulated Data');

subplot(512);
hold on;
plot(r, 'b');
plot(r_smth, 'r-.', 'linewidth', 1.25); grid;
ylabel('(b) r_{k}');

subplot(513);
hold on;
plot(s, 'b');
plot(s_smth, 'r-.', 'linewidth', 1.25); grid;
ylabel('(c) s_{k}');

subplot(514);
hold on;
plot(x, 'b');
plot(x_smth, 'r-.', 'linewidth', 1.25); grid;
ylabel('(d) x_{k}'); xlabel('time index');

subplot(515);
qqplot(x - x_smth);
title('QQ Plot - State Estimate', 'FontWeight', 'Normal');
ylabel('(e) input quantiles');
xlabel('standard normal quantiles');
grid;

%% supplementary functions

function y = get_posterior_mode(x_pred, C, r, r0, r1, b0, b1, vr,
    n, s, s0, s1, vs)
```

```
    M = 200;    % maximum iterations

    it = zeros(1, M);
    f = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)
        f(i) = it(i) - x_pred - C * (r1 * vs * (r - r0 - r1 *
    x_pred) + s1 * vr * (s - s0 - s1 * x_pred) + ...
            vr * vs * b1 * (n - (1 / (1 + exp((-1) * (b0 + b1 *
    it(i)))))));
        df(i) = 1 + C * vr * vs * (b1 ^ 2) * exp(b0 + b1 * it(i))
     / ((1 + exp(b0 + b1 * it(i))) ^ 2);
        it(i + 1) = it(i) - f(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
           y = it(i + 1);
           return;
        end
    end

    error('Newton-Raphson failed to converge.');

end

function y = get_maximum_variance(z, r0, r1, W, x_smth, K)

    y = (z * z' + K * (r0 ^ 2) + (r1 ^ 2) * sum(W) ...
                - 2 * r0 * sum(z) - 2 * r1 * dot(x_smth, z) + 2 *
     r0 * r1 * sum(x_smth)) / K;

end

function y = get_linear_parameters(x_smth, W, z, K)

    y = [K sum(x_smth); sum(x_smth) sum(W)] \ [sum(z); sum(z .*
    x_smth)];

end
```

10.3.2 Simulated Data Example

```
load('data_one_bin_two_cont.mat');

base_prob = sum(n) / length(n);

%% parameters

M = 1e6;    % maximum iterations
```

```
m = 1;
tol = 1e-8; % convergence criteria

b0 = zeros(1, M);   % binary GLM model
b1 = zeros(1, M);

r0 = zeros(1, M);   % continuous model
r1 = zeros(1, M);
vr = zeros(1, M);   % continuous model noise variance (1)

s0 = zeros(1, M);   % continuous model
s1 = zeros(1, M);
vs = zeros(1, M);   % continuous model noise variance (2)

ve = zeros(1, M);   % process noise variance
rho = zeros(1, M);  % random walk forgetting factor
alpha = zeros(1, M);    % external input gain parameter

K = length(n);

x_pred = zeros(1, K);
v_pred = zeros(1, K);

x_updt = zeros(1, K);
v_updt = zeros(1, K);

x_smth = zeros(1, K);
v_smth = zeros(1, K);

p_updt = zeros(1, K);

A = zeros(1, K);
W = zeros(1, K);
CW = zeros(1, K);
C = zeros(1, K);

%% initial guesses

b0(1) = log(base_prob / (1 - base_prob));
b1(1) = 1;
r0(1) = r(1); % guess it's the first value of r
r1(1)  = 0.5;
s0(1) = s(1);
s1(1) = 1;
vr(1) = 0.05;
vs(1) = 0.05;
ve(1) = 0.05;
rho(1) = 1;
alpha(1) = 0.5;

%% main function

for m = 1:M
```

```
    for k = 1:K

        if (k == 1)
            x_pred(k) = x_smth(1);
            v_pred(k) = ve(m) + ve(m);
        else
            x_pred(k) = rho(m) * x_updt(k - 1) + alpha(m) * I(k);
            v_pred(k) = (rho(m) ^ 2) * v_updt(k - 1) + ve(m);
        end

        C(k) = v_pred(k) / (vr(m) * vs(m) + v_pred(k) * ((r1(m) ^
     2) * vs(m) + (s1(m) ^ 2) * vr(m)));
        x_updt(k) = get_posterior_mode(x_pred(k), C(k), r(k), r0(
    m), r1(m), b0(m), b1(m), vr(m), n(k), s(k), s0(m), s1(m), vs(
    m));

        p_updt(k) = 1 / (1 + exp((-1) * (b0(m) + b1(m) * x_updt(k
    ))));
        v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(m))
    + ((s1(m) ^ 2) / vs(m)) + (b1(m) ^ 2) * p_updt(k) * (1 -
    p_updt(k)));
    end

    x_smth(K) = x_updt(K);
    v_smth(K) = v_updt(K);
    W(K) = v_smth(K) + (x_smth(K) ^ 2);

    A(1:(end - 1)) = rho(m) * v_updt(1:(end - 1)) ./ v_pred(2:end
    );

    for k = (K - 1):(-1):1
       x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
     1));
       v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
    v_pred(k + 1));

       CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
       W(k) = v_smth(k) + (x_smth(k) ^ 2);
    end

    prev = [alpha(m) r0(m) r1(m) ve(m) vr(m) rho(m) s0(m) s1(m)
    vs(m)];

    R = get_linear_parameters(x_smth, W, r, K);
    S = get_linear_parameters(x_smth, W, s, K);

    Q = [sum(W(1:end - 1)) (I(2:end) * x_smth(1:(end - 1))'); ...
        (I(2:end) * x_smth(1:(end - 1))') (I * I')] \ [sum(CW); (
    I(2:end) * x_smth(2:end)')];

    b0(m + 1) = log(base_prob / (1 - base_prob));
    b1(m + 1) = 1;

    rho(m + 1) = Q(1, 1);
```

```
    alpha(m + 1) = Q(2, 1);

    ve(m + 1) = (sum(W(2:end)) + (rho(m + 1) ^ 2) * sum(W(1:(end
    - 1))) - 2 * rho(m + 1) * sum(CW) - ...
        2 * alpha(m + 1) * (I(2:end) * x_smth(2:end)') + 2 *
    alpha(m + 1) * rho(m + 1) * (I(2:end) * x_smth(1:(end - 1))')
     + ...
        (alpha(m + 1) ^ 2) * (I * I')) / K;

    r0(m + 1) = R(1, 1);
    r1(m + 1) = R(2, 1);

    s0(m + 1) = S(1, 1);
    s1(m + 1) = S(2, 1);

    vr(m + 1) = get_maximum_variance(r, r0(m + 1), r1(m + 1), W,
    x_smth, K);
    vs(m + 1) = get_maximum_variance(s, s0(m + 1), s1(m + 1), W,
    x_smth, K);

    next = [alpha(m + 1) r0(m + 1) r1(m + 1) ve(m + 1) vr(m + 1)
    rho(m + 1) s0(m + 1) s1(m + 1) vs(m +1)];

    mean_dev = mean(abs(next - prev));

    if mean_dev < tol
        fprintf('Converged at m = %d\n\n', m);
        break;
    else
        fprintf('m = %d\nr0 = %.18f\nr1 = %.18f\nvr = %.18f\n\ns0
     = %.18f\ns1 = %.18f\nvs = %.18f\n\nve = %.18f\nrho = %.18f\
    nalpha = %.18f\n\n', ...
        m + 1, r0(m + 1), r1(m + 1), vr(m + 1), s0(m + 1), s1(m +
     1), vs(m + 1), ve(m + 1), rho(m + 1), alpha(m + 1));

        x_pred = zeros(1, K);
        v_pred = zeros(1, K);

        x_updt = zeros(1, K);
        v_updt = zeros(1, K);

        x_smth(2:end) = zeros(1, K - 1);    % x_smth(1) needed
    for next iteration
        v_smth = zeros(1, K);

        p_updt = zeros(1, K);

        A = zeros(1, K);
        W = zeros(1, K);
        CW = zeros(1, K);
        C = zeros(1, K);
    end
end
```

```
%% calculate confidence limits

p_smth = 1 ./ (1 + exp((-1) * (b0(m) + b1(m) * x_smth)));  % mode
    , lower and upper confidence limits for binary distribution
r_smth = r0(m) + r1(m) * x_smth;
s_smth = s0(m) + s1(m) * x_smth;

%% plot graphs

figure;
subplot(511);
hold on;
plot(p, 'b');
plot(p_smth, 'r-.', 'linewidth', 1.25); grid;
plot(find(n == 0) - 1, 1.2 * max(p_smth) * ones(length(find(n ==
    0))), 'ks', 'MarkerFaceColor', 'k', 'MarkerSize', 4);
plot(find(n == 1) - 1, 1.2 * max(p_smth) * ones(length(find(n ==
    1))), 'gs', 'MarkerFaceColor', 'g', 'MarkerSize', 4);
ylabel('(a) p_{k}'); ylim([0 0.17]);
title('State Estimation with Simulated Data');

subplot(512);
hold on;
plot(r, 'b');
plot(r_smth, 'r-.', 'linewidth', 1.25); grid;
ylabel('(b) r_{k}');

subplot(513);
hold on;
plot(s, 'b');
plot(s_smth, 'r-.', 'linewidth', 1.25); grid;
ylabel('(c) s_{k}');

subplot(514);
hold on;
plot(x, 'b');
plot(x_smth, 'r-.', 'linewidth', 1.25); grid;
plot(find(I == 0) - 1, (-8) * ones(length(find(I == 0))), 'ks', '
    MarkerFaceColor', 'k', 'MarkerSize', 4);
plot(find(I == 1) - 1, (-8) * ones(length(find(I == 1))), 'cs', '
    MarkerFaceColor', 'c', 'MarkerSize', 4);
ylabel('(d) x_{k}'); xlabel('time index');

subplot(515);
qqplot(x - x_smth);
title('QQ Plot - State Estimate', 'FontWeight', 'Normal');
ylabel('(e) input quantiles');
xlabel('standard normal quantiles');
grid;

%% supplementary functions

function y = get_posterior_mode(x_pred, C, r, r0, r1, b0, b1, vr,
    n, s, s0, s1, vs)
```

```
    M = 200;    % maximum iterations

    it = zeros(1, M);
    f = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)
        f(i) = it(i) - x_pred - C * (r1 * vs * (r - r0 - r1 *
    x_pred) + s1 * vr * (s - s0 - s1 * x_pred) + ...
            vr * vs * b1 * (n - (1 / (1 + exp((-1) * (b0 + b1 *
    it(i)))))));
        df(i) = 1 + C * vr * vs * (b1 ^ 2) * exp(b0 + b1 * it(i))
     / ((1 + exp(b0 + b1 * it(i))) ^ 2);
        it(i + 1) = it(i) - f(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
           y = it(i + 1);
           return;
        end
    end

    error('Newton-Raphson failed to converge.');

end

function y = get_maximum_variance(z, r0, r1, W, x_smth, K)

    y = (z * z' + K * (r0 ^ 2) + (r1 ^ 2) * sum(W) ...
                - 2 * r0 * sum(z) - 2 * r1 * dot(x_smth, z) + 2 *
     r0 * r1 * sum(x_smth)) / K;

end

function y = get_linear_parameters(x_smth, W, z, K)

    y = [K sum(x_smth); sum(x_smth) sum(W)] \ [sum(z); sum(z .*
    x_smth)];

end
```

10.3.3 Experimental Data Example (αI_k Excluded)

```
load('expm_data_one_bin_two_cont_no_extern_stim.mat');

min_scr_thresh = 0.015;
min_scr_prom = min_scr_thresh;
fs = 2;

t = (0:(length(phasic) - 1)) / fs;
```

```
ph = phasic;
tn = tonic;
x_orig = y;

[pks, locs] = findpeaks(ph, 'MinPeakHeight', min_scr_thresh, '
    MinPeakProminence', min_scr_prom);

r = interp1([1 locs length(ph)], log([ph(1) pks ph(end)]), 1:
    length(ph), 'cubic');
s = tn;
n = zeros(1, length(r));
I = zeros(1, length(r));
n(locs) = 1;

base_prob = sum(n) / length(n);

std_s = std(s);
std_r = std(r);

s = s / std_s;
r = r / std_r;

%% parameters

M = 5e5;    % maximum iterations
tol = 1e-8; % convergence criteria

b0 = zeros(1, M);   % binary GLM model
b1 = zeros(1, M);

r0 = zeros(1, M);   % continuous model
r1 = zeros(1, M);
vr = zeros(1, M);   % continuous model noise variance (1)

s0 = zeros(1, M);   % continuous model
s1 = zeros(1, M);
vs = zeros(1, M);   % continuous model noise variance (2)

ve = zeros(1, M);   % process noise variance
rho = zeros(1, M);  % random walk forgetting factor

K = length(n);

x_pred = zeros(1, K);
v_pred = zeros(1, K);

x_updt = zeros(1, K);
v_updt = zeros(1, K);

x_smth = zeros(1, K);
v_smth = zeros(1, K);

p_updt = zeros(1, K);
```

```
A = zeros(1, K);
W = zeros(1, K);
CW = zeros(1, K);
C = zeros(1, K);

%% initial guesses

b0(1) = log(base_prob / (1 - base_prob));
b1(1) = 1;
r0(1) = r(1); % guess it's the first value of r
r1(1)  = 1;
s0(1) = s(1);
s1(1) = 1;
vr(1) = 0.05;
vs(1) = 0.05;
ve(1) = 0.05;
rho(1) = 1;

%% main function

for m = 1:M

    for k = 1:K

        if (k == 1)
            x_pred(k) = x_smth(1);
            v_pred(k) = ve(m) + ve(m);
        else
            x_pred(k) = rho(m) * x_updt(k - 1);
            v_pred(k) = (rho(m) ^ 2) * v_updt(k - 1) + ve(m);
        end

        C(k) = v_pred(k) / (vr(m) * vs(m) + v_pred(k) * ((r1(m) ^
     2) * vs(m) + (s1(m) ^ 2) * vr(m)));
        x_updt(k) = get_posterior_mode(x_pred(k), C(k), r(k), r0(
    m), r1(m), b0(m), b1(m), vr(m), n(k), s(k), s0(m), s1(m), vs(
    m));

        p_updt(k) = 1 / (1 + exp((-1) * (b0(m) + b1(m) * x_updt(k
    ))));
        v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(m))
    + ((s1(m) ^ 2) / vs(m)) + (b1(m) ^ 2) * p_updt(k) * (1 -
    p_updt(k)));
    end

    x_smth(K) = x_updt(K);
    v_smth(K) = v_updt(K);
    W(K) = v_smth(K) + (x_smth(K) ^ 2);

    A(1:(end - 1)) = rho(m) * v_updt(1:(end - 1)) ./ v_pred(2:end
    );

    for k = (K - 1):(-1):1
```

```
    x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
 1));
    v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
v_pred(k + 1));

    CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
    W(k) = v_smth(k) + (x_smth(k) ^ 2);
end

if (m < M)

    R = get_linear_parameters(x_smth, W, r, K);
    S = get_linear_parameters(x_smth, W, s, K);

    b0(m + 1) = log(base_prob / (1 - base_prob));
    b1(m + 1) = 1;

    rho(m + 1) = sum(CW) / sum(W(1:end - 1));

    ve(m + 1) = (sum(W(2:end)) + (rho(m + 1) ^ 2) * sum(W(1:(
end - 1))) - 2 * rho(m + 1) * sum(CW)) / K;

    if (abs(get_maximum_variance(r, R(1, 1), R(2, 1), W,
x_smth, K) - get_maximum_variance(s, S(1, 1), S(2, 1), W,
x_smth, K)) > 0.1)   % overfitting check
        r0(m + 1) = r0(m);
        r1(m + 1) = r1(m);

        s0(m + 1) = s0(m);
        s1(m + 1) = s1(m);

        vr(m + 1) = vr(m);
        vs(m + 1) = vs(m);

        mean_dev = mean(abs([ve(m + 1) rho(m + 1)] - [ve(m)
rho(m)]));
    else
        r0(m + 1) = R(1, 1);
        r1(m + 1) = R(2, 1);

        s0(m + 1) = S(1, 1);
        s1(m + 1) = S(2, 1);

        vr(m + 1) = get_maximum_variance(r, r0(m + 1), r1(m +
 1), W, x_smth, K);
        vs(m + 1) = get_maximum_variance(s, s0(m + 1), s1(m +
 1), W, x_smth, K);

        mean_dev = mean(abs([r0(m + 1) r1(m + 1) ve(m + 1) vr
(m + 1) rho(m + 1) s0(m + 1) s1(m + 1) vs(m + 1)] - ...
            [r0(m) r1(m) ve(m) vr(m) rho(m) s0(m) s1(m) vs(m)
]));
    end
```

```
        if mean_dev < tol
            fprintf('Converged at m = %d\n\n', m);
            break;
        else
            fprintf('m = %d\nb0 = %.18f\nb1 = %.18f\n\nr0 = %.18f
    \nr1 = %.18f\nvr = %.18f\n\ns0 = %.18f\ns1 = %.18f\nvs = %.18
    f\n\nve = %.18f\nrho = %.18f\n\n', ...
            m + 1, b0(m + 1), b1(m + 1), r0(m + 1), r1(m + 1), vr
    (m + 1), s0(m + 1), s1(m + 1), vs(m + 1), ve(m + 1), rho(m +
    1));

            x_pred = zeros(1, K);
            v_pred = zeros(1, K);

            x_updt = zeros(1, K);
            v_updt = zeros(1, K);

            x_smth(2:end) = zeros(1, K - 1);    % x_smth(1)
    needed for next iteration
            v_smth = zeros(1, K);

            p_updt = zeros(1, K);

            A = zeros(1, K);
            W = zeros(1, K);
            CW = zeros(1, K);
            C = zeros(1, K);
        end
    end
end

%% calculate confidence limits

p_smth = 1 ./ (1 + exp((-1) * (b0(m) + b1(m) * x_smth)));  % mode
    , lower and upper confidence limits for binary distribution
r_smth = exp(std_r * (r0(m) + r1(m) * x_smth));

s_smth = (s0(m) + s1(m) * x_smth) * std_s;

lcl_x = norminv(0.025, x_smth, sqrt(v_smth));
ucl_x = norminv(0.975, x_smth, sqrt(v_smth));

lcl_p = zeros(1, K);
ucl_p = zeros(1, K);

for k = 1:K
    [lcl_p(k), ucl_p(k)] = get_pk_conf_lims(v_smth(k), b0(m),
    x_smth(k));
end

certainty = 1 - normcdf(prctile(x_smth, 50) * ones(1, length(
    x_smth)), x_smth, sqrt(v_smth));

%% plot graphs
```

```
disp('Plotting...');

xp_fs_plot = 4;

index = (0:(K - 1));
t_index = index / fs;
r_index = ((K - 1):(-1):0) / fs;
transp = 0.3;

subplot(611);
hold on;
plot(t_index, x_orig, 'k', 'linewidth', 1.25);
plot(find(n == 0) / fs, 3.7 * ones(length(find(n == 0))), 'ks', '
    MarkerFaceColor', 'k', 'MarkerSize', 5);
plot(find(n == 1) / fs, 3.7 * ones(length(find(n == 1))), 'gs', '
    MarkerFaceColor', 'g', 'MarkerSize', 5);
ylim([0 4]); yl = ylim;

patch([xp(1), xp(2), xp(2), xp(1)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'r', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(2), xp(3), xp(3), xp(2)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'g', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(3), xp(4), xp(4), xp(3)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], [1 0.647059 0], 'FaceAlpha', 0.2, 'EdgeColor', '
    none');
patch([xp(4), xp(5), xp(5), xp(4)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(5), xp(6), xp(6), xp(5)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'y', 'FaceAlpha', 0.2, 'EdgeColor', 'none');

ylabel({'(a) skin cond.', '(\mu S)'}); grid; xlim([0 (xp(6) /
    xp_fs_plot)]);
set(gca,'xticklabel', []);
title('State Estimation with Experimental Data');

subplot(612);
hold on;

plot(t_index, r_smth, ':', 'color', [0 0.3 0], 'linewidth', 1.5);
plot(t_index, exp(r * std_r), 'color', [0 0.9 0], 'linewidth',
    1.5);
grid;

xlim([0 (xp(6) / xp_fs_plot)]);
ylim([(min([exp(r * std_r) r_smth]) - 0.25) (0.25 +  max([exp(r *
     std_r) r_smth]))]);  yl = ylim;

patch([xp(1), xp(2), xp(2), xp(1)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'r', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(2), xp(3), xp(3), xp(2)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'g', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(3), xp(4), xp(4), xp(3)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], [1 0.647059 0], 'FaceAlpha', 0.2, 'EdgeColor', '
    none');
```

```
patch([xp(4), xp(5), xp(5), xp(4)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(5), xp(6), xp(6), xp(5)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'y', 'FaceAlpha', 0.2, 'EdgeColor', 'none');

set(gca,'xticklabel', []);
ylabel('(b) phasic');

subplot(613);
hold on;

plot(t_index, s_smth, ':', 'color', [0.5 (25 / 255) (66 / 255)],
    'linewidth', 1.5);
plot(t_index, s * std_s, 'color', [1 0.5 (179 / 255)], 'linewidth
    ', 1.5); grid;

xlim([0 (xp(6) / xp_fs_plot)]);
ylim([(min([(s * std_s) s_smth]) - 0.25) (0.25 + max([(s * std_s)
     s_smth]))]); yl = ylim;

patch([xp(1), xp(2), xp(2), xp(1)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'r', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(2), xp(3), xp(3), xp(2)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'g', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(3), xp(4), xp(4), xp(3)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], [1 0.647059 0], 'FaceAlpha', 0.2, 'EdgeColor', '
    none');
patch([xp(4), xp(5), xp(5), xp(4)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(5), xp(6), xp(6), xp(5)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'y', 'FaceAlpha', 0.2, 'EdgeColor', 'none');

set(gca,'xticklabel', []);
ylabel('(c) tonic');

subplot(614);
hold on;
plot(t_index, x_smth, 'color', 'b', 'linewidth', 1.25); grid;
fill([t_index, r_index], [lcl_x fliplr(ucl_x)], 'c', 'EdgeColor',
     'none', 'FaceAlpha', 0.5);
ylim([(min(x_smth) - 0.25) (0.25 + max(x_smth))]); yl = ylim;

patch([xp(1), xp(2), xp(2), xp(1)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'r', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(2), xp(3), xp(3), xp(2)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'g', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(3), xp(4), xp(4), xp(3)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], [1 0.647059 0], 'FaceAlpha', 0.2, 'EdgeColor', '
    none');
patch([xp(4), xp(5), xp(5), xp(4)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(5), xp(6), xp(6), xp(5)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'y', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
```

```
xlim([0 (xp(6) / xp_fs_plot)]);

set(gca,'xticklabel', []);
ylabel('(d) state (x_{k})');

subplot(615);
hold on;
plot(t_index, p_smth, 'r', 'linewidth', 1.5); grid;
fill([t_index, r_index], [lcl_p fliplr(ucl_p)], [1, 0, (127 /
    255)], 'EdgeColor', 'none', 'FaceAlpha', 0.3);

xlim([0 (xp(6) / xp_fs_plot)]);
ylim([0 (max(p_smth) * 1.5)]); yl = ylim;

patch([xp(1), xp(2), xp(2), xp(1)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'r', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(2), xp(3), xp(3), xp(2)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'g', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(3), xp(4), xp(4), xp(3)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], [1 0.647059 0], 'FaceAlpha', 0.2, 'EdgeColor', '
    none');
patch([xp(4), xp(5), xp(5), xp(4)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(5), xp(6), xp(6), xp(5)] / xp_fs_plot, [yl(1) yl(1) yl
    (2) yl(2)], 'y', 'FaceAlpha', 0.2, 'EdgeColor', 'none');

set(gca,'xticklabel',[]);
ylabel({'(e) probability', '(p_{k})'}, 'FontSize', 11);

subplot(616);
hold on;
v1 = [0 0.9; t(end) 0.9; t(end) 1; 0 1];
c1 = [1 (220 / 255) (220 / 255); 1 (220 / 255) (220 / 255); 1 0
    0; 1 0 0];
faces1 = [1 2 3 4];

patch('Faces', faces1, 'Vertices', v1, 'FaceVertexCData', c1, '
    FaceColor', 'interp', ...
    'EdgeColor', 'none', 'FaceAlpha', 0.7);

v2 = [0 0; t(end) 0; t(end) 0.1; 0 0.1];
c2 = [0 0.8 0; 0 0.8 0; (204 / 255) 1 (204 / 255); (204 / 255) 1
    (204 / 255)];
faces2 = [1 2 3 4];

patch('Faces', faces2, 'Vertices', v2, 'FaceVertexCData', c2, '
    FaceColor', 'interp', ...
    'EdgeColor', 'none', 'FaceAlpha', 0.7);

plot(t, certainty, 'color', [(138 / 255) (43 / 255) (226 / 255)],
     'linewidth', 1.5); grid;
xlim([0 (xp(6) / xp_fs_plot)]);

xlabel('time (s)');
```

```
ylabel('(f) HAI');

%% supplementary functions

function y = get_posterior_mode(x_pred, C, r, r0, r1, b0, b1, vr,
     n, s, s0, s1, vs)

    M = 200;    % maximum iterations

    it = zeros(1, M);
    f = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)
        f(i) = it(i) - x_pred - C * (r1 * vs * (r - r0 - r1 *
    x_pred) + s1 * vr * (s - s0 - s1 * x_pred) + ...
            vr * vs * b1 * (n - (1 / (1 + exp((-1) * (b0 + b1 *
    it(i)))))));
        df(i) = 1 + C * vr * vs * (b1 ^ 2) * exp(b0 + b1 * it(i))
     / ((1 + exp(b0 + b1 * it(i))) ^ 2);
        it(i + 1) = it(i) - f(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
           y = it(i + 1);
           return;
        end
    end

    error('Newton-Raphson failed to converge.');

end

function [lcl, ucl] = get_pk_conf_lims(v, b0, x)

    p = (1e-4:1e-4:1);

    fp = cumtrapz(p, 1 ./ (sqrt(2 * pi * v) * p .* (1 - p)) .*
    ...
        exp(((-1) / (2 * v))* (log(p ./ ((1 - p) * exp(b0))) - x)
     .^ 2));

    n = find(fp <= 0.975);
    m = find(fp < 0.025);

    ucl = p(n(end));
    lcl = p(m(end));
end


function y = get_maximum_variance(z, r0, r1, W, x_smth, K)

    y = (z * z' + K * (r0 ^ 2) + (r1 ^ 2) * sum(W) ...
```

```
                - 2 * r0 * sum(z) - 2 * r1 * dot(x_smth, z) + 2 *
    r0 * r1 * sum(x_smth)) / K;

end

function y = get_linear_parameters(x_smth, W, z, K)

    y = [K sum(x_smth); sum(x_smth) sum(W)] \ [sum(z); sum(z .*
    x_smth)];

end
```

10.3.4 Experimental Data Example

```
load('expm_data_one_bin_two_cont.mat');

min_scr_thresh = 0.015;
min_scr_prom = min_scr_thresh;
fs = 4;
epoch = 10;

subj = 1;

stim = s_data.aug_stim;
ph = s_data.ph;
tn = s_data.tn;

[pks, locs] = findpeaks(ph, 'MinPeakHeight', min_scr_thresh, '
    MinPeakProminence', min_scr_prom);
r = interp1([1 locs length(ph)], log([ph(find(ph > 0, 1)) pks ph(
    end)]), 1:length(ph), 'cubic');

s = tn;
n = zeros(1, length(r));
I = zeros(1, length(r));

n(locs) = 1;
I(stim) = 1;

std_s = std(s);
std_r = std(r);

s = s / std_s;
r = r / std_r;

%% parameters

M = 5e5;    % maximum iterations
tol = 1e-8; % convergence criteria

b0 = zeros(1, M);   % binary GLM model
b1 = zeros(1, M);
```

```
r0 = zeros(1, M);   % continuous model
r1 = zeros(1, M);
vr = zeros(1, M);   % continuous model noise variance (1)

s0 = zeros(1, M);   % continuous model
s1 = zeros(1, M);
vs = zeros(1, M);   % continuous model noise variance (2)

ve = zeros(1, M);   % process noise variance
rho = zeros(1, M);  % random walk forgetting factor
alpha = zeros(1, M);    % input gain parameter

K = length(n);

x_pred = zeros(1, K);
v_pred = zeros(1, K);

x_updt = zeros(1, K);
v_updt = zeros(1, K);

x_smth = zeros(1, K);
v_smth = zeros(1, K);

p_updt = zeros(1, K);
p_smth = zeros(1, K);

A = zeros(1, K);
W = zeros(1, K);
CW = zeros(1, K);
C = zeros(1, K);

%% initial guesses

base_prob = sum(n) / length(n);
b0(1) = log(base_prob / (1 - base_prob));
b1(1) = 1;
r0(1) = r(1); % guess it's the first value of r
r1(1)  = 1;
s0(1) = s(1);
s1(1) = 1;
vr(1) = 0.05;
vs(1) = 0.05;
ve(1) = 0.05;
rho(1) = 1;
alpha(1) = 0.5;

%% main function

for m = 1:M

    for k = 1:K

        if (k == 1)
```

```
            x_pred(k) = x_smth(1);
            v_pred(k) = ve(m) + ve(m);
        else
            x_pred(k) = rho(m) * x_updt(k - 1) + alpha(m) * I(k);
            v_pred(k) = (rho(m) ^ 2) * v_updt(k - 1) + ve(m);
        end

        C(k) = v_pred(k) / (vr(m) * vs(m) + v_pred(k) * ((r1(m) ^
 2) * vs(m) + (s1(m) ^ 2) * vr(m)));
        x_updt(k) = get_posterior_mode(x_pred(k), C(k), r(k), r0(
m), r1(m), b0(m), b1(m), vr(m), n(k), s(k), s0(m), s1(m), vs(
m));

        p_updt(k) = 1 / (1 + exp((-1) * (b0(m) + b1(m) * x_updt(k
))));
        v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(m))
+ ((s1(m) ^ 2) / vs(m)) + (b1(m) ^ 2) * p_updt(k) * (1 -
p_updt(k)));
    end

    x_smth(K) = x_updt(K);
    v_smth(K) = v_updt(K);
    W(K) = v_smth(K) + (x_smth(K) ^ 2);

    A(1:(end - 1)) = rho(m) * v_updt(1:(end - 1)) ./ v_pred(2:end
    );

    for k = (K - 1):(-1):1
        x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
 1));
        v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
v_pred(k + 1));

        CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
        W(k) = v_smth(k) + (x_smth(k) ^ 2);
    end

    if (m < M)

        R = get_linear_parameters(x_smth, W, r, K);
        S = get_linear_parameters(x_smth, W, s, K);

        Q = [sum(W(1:end - 1)) (I(2:end) * x_smth(1:(end - 1))');
 ...
            (I(2:end) * x_smth(1:(end - 1))') (I * I')] \ [sum(CW
); (I(2:end) * x_smth(2:end)')];

        b0(m + 1) = log(base_prob / (1 - base_prob));
        b1(m + 1) = 1;

        rho(m + 1) = Q(1, 1);

        if (Q(2, 1) < 0)
            alpha(m + 1) = alpha(m);
```

```
        else
            alpha(m + 1) = Q(2, 1);
        end

        ve(m + 1) = (sum(W(2:end)) + (rho(m + 1) ^ 2) * sum(W(1:(
    end - 1))) - 2 * rho(m + 1) * sum(CW) - ...
            2 * alpha(m + 1) * (I(2:end) * x_smth(2:end)') + 2 *
    alpha(m + 1) * rho(m + 1) * (I(2:end) * x_smth(1:(end - 1))')
     + ...
            (alpha(m + 1) ^ 2) * (I * I')) / K;

        if (abs(get_maximum_variance(r, R(1, 1), R(2, 1), W,
    x_smth, K) - get_maximum_variance(s, S(1, 1), S(2, 1), W,
    x_smth, K)) > 0.1)    % overfitting check
            r0(m + 1) = r0(m);
            r1(m + 1) = r1(m);

            s0(m + 1) = s0(m);
            s1(m + 1) = s1(m);

            vr(m + 1) = vr(m);
            vs(m + 1) = vs(m);

            mean_dev = mean(abs([ve(m + 1) rho(m + 1) alpha(m +
    1)] - [ve(m) rho(m) alpha(m)]));
        else
            r0(m + 1) = R(1, 1);
            r1(m + 1) = R(2, 1);

            s0(m + 1) = S(1, 1);
            s1(m + 1) = S(2, 1);

            vr(m + 1) = get_maximum_variance(r, r0(m + 1), r1(m +
     1), W, x_smth, K);
            vs(m + 1) = get_maximum_variance(s, s0(m + 1), s1(m +
     1), W, x_smth, K);

            mean_dev = mean(abs([r0(m + 1) r1(m + 1) ve(m + 1) vr
    (m + 1) rho(m + 1) alpha(m + 1) s0(m + 1) s1(m + 1) vs(m + 1)
    ] - ...
                [r0(m) r1(m) ve(m) vr(m) rho(m) alpha(m) s0(m) s1
    (m) vs(m)]));
        end

        if mean_dev < tol
            fprintf('Converged at m = %d\n\n', m);
            break;
        else
            fprintf('m = %d\nb0 = %.18f\nb1 = %.18f\n\nr0 = %.18f
    \nr1 = %.18f\nvr = %.18f\n\ns0 = %.18f\ns1 = %.18f\nvs = %.18
    f\n\nve = %.18f\nrho = %.18f\nalpha = %.18f\n\n', ...
            m + 1, b0(m + 1), b1(m + 1), r0(m + 1), r1(m + 1), vr
    (m + 1), s0(m + 1), s1(m + 1), vs(m + 1), ve(m + 1), rho(m +
    1), alpha(m + 1));
```

```
            x_pred = zeros(1, K);
            v_pred = zeros(1, K);

            x_updt = zeros(1, K);
            v_updt = zeros(1, K);

            x_smth(2:end) = zeros(1, K - 1);    % x_smth(1)
    needed for next iteration
            v_smth = zeros(1, K);

            p_updt = zeros(1, K);
            p_smth = zeros(1, K);

            A = zeros(1, K);
            W = zeros(1, K);
            CW = zeros(1, K);
            C = zeros(1, K);
        end
    end
end

%% calculate confidence limits

fp_mode = 1 ./ (1 + exp((-1) * (b0(m) + b1(m) * x_smth)));  %
    mode, lower and upper confidence limits for binary
    distribution
lcl_fp = zeros(1, K);
ucl_fp = zeros(1, K);

r_smth = exp((r0(m) + r1(m) * x_smth) * std_r);
s_smth = (s0(m) + s1(m) * x_smth) * std_s;

skn_avg = get_trial_averages(s_data, x_smth, epoch, fs, 'skn');
x_avg = get_trial_averages(s_data, x_smth, epoch, fs, 'x_smth');

t_epoch = ((-1):(1 / fs):(epoch - 1 - (1 / fs)));
tr_epoch = ((epoch - 1 - (1 / fs)):(-1 / fs):(-1));

%% plot graphs
disp('Plotting...(you may need to press the Enter key again)');

index = (0:(K - 1));
t_index = index / fs;
r_index = ((K - 1):(-1):0); % reverse index
transp = 0.3;

subplot(611);
plot(t_index, s_data.x, 'color', [(102 / 255) 0 (204 / 255)]);
ylabel('(a) z_{k}'); grid; xlim([0 t_index(end)]);
set(gca,'xticklabel', []);
ylim([(min(s_data.x) - 0.1) (max(s_data.x) + 0.1)]);
title('State Estimation with Experimental Data');
```

```
subplot(612);
hold on;
plot(find(n == 0) / fs, max(fp_mode) * 1.3 * ones(length(find(n
    == 0))), 'ks', 'MarkerFaceColor', 'k', 'MarkerSize', 3);
plot(find(n == 1) / fs, max(fp_mode) * 1.3 * ones(length(find(n
    == 1))), 'gs', 'MarkerFaceColor', 'g', 'MarkerSize', 3);
plot(t_index, fp_mode, 'r');

ylabel('(b) p_{k}');
xlim([0 t_index(end)]); ylim([0 (max(fp_mode) * 1.5)]); grid;
set(gca,'xticklabel', []);

subplot(613);
hold on;
plot(t_index, r_smth, ':', 'color', [0 0.3 0], 'linewidth', 1.5);
plot(t_index, exp(r * std_r), 'color', [0 0.9 0]);

ylabel('(c) e^{r_{k}}'); grid;
xlim([0 t_index(end)]);
set(gca,'xticklabel', []);

subplot(614);
hold on;
plot(t_index, s_smth, ':', 'color', [0.5 (25 / 255) (66 / 255)],
    'linewidth', 1.5);
plot(t_index, s * std_s, 'color', [1 0.5 (179 / 255)]);

ylabel('(d) s_{k}');
xlim([0 t_index(end)]); grid;
set(gca,'xticklabel', []);

subplot(615);
hold on;
plot(t_index, x_smth, 'color', 'b');
plot(find(I == 0) / fs, (min(x_smth) - 0.5) * ones(length(find(I
    == 0))), 'ks', 'MarkerFaceColor', 'k', 'MarkerSize', 3);
plot(find(I == 1) / fs, (min(x_smth) - 0.5) * ones(length(find(I
    == 1))), 'cs', 'MarkerFaceColor', 'c', 'MarkerSize', 3);

ylabel('(e) x_{k}'); ylim([(min(x_smth) - 1) (max(x_smth) + 1)]);
xlim([0 t_index(end)]); grid; xlabel('Time (s)');

subplot(6, 2, 11);
hold on;
plot(t_epoch, skn_avg(1, :), 'r', 'linewidth', 1.5);
fill([t_epoch, tr_epoch], [skn_avg(2, :) fliplr(skn_avg(3, :))],
    'r', 'EdgeColor', 'none', 'FaceAlpha', 0.2);

plot(t_epoch, skn_avg(4, :), 'm', 'linewidth', 1.5);
fill([t_epoch, tr_epoch], [skn_avg(5, :) fliplr(skn_avg(6, :))],
    'm', 'EdgeColor', 'none', 'FaceAlpha', 0.2);

plot(t_epoch, skn_avg(7, :), 'color', [0 0.8 0], 'linewidth',
    1.5);
```

```
fill([t_epoch, tr_epoch], [skn_avg(8, :) fliplr(skn_avg(9, :))],
    'g', 'EdgeColor', 'none', 'FaceAlpha', 0.2);
xlim([t_epoch(1) t_epoch(end)]);
ylim([(min(min(skn_avg)) - 0.5) (max(max(skn_avg)) + 0.5)]);

grid;
xlabel('Time (s)'); ylabel('(f) z_{k}');

subplot(6, 2, 12);
hold on;
plot(t_epoch, x_avg(1, :), 'r', 'linewidth', 1.5);
fill([t_epoch, tr_epoch], [x_avg(2, :) fliplr(x_avg(3, :))], 'r',
     'EdgeColor', 'none', 'FaceAlpha', 0.2);

plot(t_epoch, x_avg(4, :), 'm', 'linewidth', 1.5);
fill([t_epoch, tr_epoch], [x_avg(5, :) fliplr(x_avg(6, :))], 'm',
     'EdgeColor', 'none', 'FaceAlpha', 0.2);

plot(t_epoch, x_avg(7, :), 'color', [0 0.8 0], 'linewidth', 1.5);
fill([t_epoch, tr_epoch], [x_avg(8, :) fliplr(x_avg(9, :))], 'g',
     'EdgeColor', 'none', 'FaceAlpha', 0.2);
xlim([t_epoch(1) t_epoch(end)]);
ylim([(min(min(x_avg)) - 0.2) (max(max(x_avg)) + 0.2)]);

grid;
xlabel('time (s)'); ylabel('(g) x_{k}');


%% supplementary functions

function y = get_posterior_mode(x_pred, C, r, r0, r1, b0, b1, vr,
     n, s, s0, s1, vs)

    M = 200;    % maximum iterations

    it = zeros(1, M);
    f = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)
        f(i) = it(i) - x_pred - C * (r1 * vs * (r - r0 - r1 *
    x_pred) + s1 * vr * (s - s0 - s1 * x_pred) + ...
            vr * vs * b1 * (n - (1 / (1 + exp((-1) * (b0 + b1 *
    it(i)))))));
        df(i) = 1 + C * vr * vs * (b1 ^ 2) * exp(b0 + b1 * it(i))
     / ((1 + exp(b0 + b1 * it(i))) ^ 2);
        it(i + 1) = it(i) - f(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
           y = it(i + 1);
           return;
        end
```

```
    end

    error('Newton-Raphson failed to converge.');

end


function y = get_maximum_variance(z, r0, r1, W, x_smth, K)

    y = (z * z' + K * (r0 ^ 2) + (r1 ^ 2) * sum(W) ...
                - 2 * r0 * sum(z) - 2 * r1 * dot(x_smth, z) + 2 *
     r0 * r1 * sum(x_smth)) / K;

end

function y = get_linear_parameters(x_smth, W, z, K)

    y = [K sum(x_smth); sum(x_smth) sum(W)] \ [sum(z); sum(z .*
    x_smth)];

end

function y = get_trial_averages(s, x_smth, epoch, fs, option)

    y = zeros(9, epoch * fs);

    csm_ep = zeros(length(s.csm), epoch * fs);
    csp_us_ep = zeros(length(s.csp_us), epoch * fs);
    csp_nus_ep = zeros(length(s.csp_nus), epoch * fs);

    csm = s.csm;
    csp_us = s.csp_us;
    csp_nus = s.csp_nus;

    if strcmp(option, 'x_smth')

        for j = 1:length(csm)
            csm_ep(j, :) = x_smth((s.stim(csm(j)) - fs ):(s.stim(
    csm(j)) + 9 * fs - 1));
        end

        for j = 1:length(csp_us)
            csp_us_ep(j, :) = x_smth((s.stim(csp_us(j)) - fs):(s.
    stim(csp_us(j)) + 9 * fs - 1));
        end

        for j = 1:length(csp_nus)
            csp_nus_ep(j, :) = x_smth((s.stim(csp_nus(j)) - fs):(
    s.stim(csp_nus(j)) + 9 * fs - 1));
        end

    elseif strcmp(option, 'skn')

        for j = 1:length(csm)
```

```
            csm_ep(j, :) = s.x((s.stim(csm(j)) - fs ):(s.stim(csm
    (j)) + 9 * fs - 1));
        end

        for j = 1:length(csp_us)
            csp_us_ep(j, :) = s.x((s.stim(csp_us(j)) - fs):(s.
    stim(csp_us(j)) + 9 * fs - 1));
        end

        for j = 1:length(csp_nus)
            csp_nus_ep(j, :) = s.x((s.stim(csp_nus(j)) - fs):(s.
    stim(csp_nus(j)) + 9 * fs - 1));
        end

    end

    y(1, :) = mean(csp_us_ep);
    y(2, :) = mean(csp_us_ep) + tinv(0.975, length(csp_us) - 1) *
     std(csp_us_ep) / sqrt(length(csp_us));
    y(3, :) = mean(csp_us_ep) + tinv(0.025, length(csp_us) - 1) *
     std(csp_us_ep) / sqrt(length(csp_us));

    y(4, :) = mean(csp_nus_ep);
    y(5, :) = mean(csp_nus_ep) + tinv(0.975, length(csp_nus) - 1)
     * std(csp_nus_ep) / sqrt(length(csp_nus));
    y(6, :) = mean(csp_nus_ep) + tinv(0.025, length(csp_nus) - 1)
     * std(csp_nus_ep) / sqrt(length(csp_nus));

    y(7, :) = mean(csm_ep);
    y(8, :) = mean(csm_ep) + tinv(0.975, length(csm) - 1) * std(
    csm_ep) / sqrt(length(csm));
    y(9, :) = mean(csm_ep) + tinv(0.025, length(csm) - 1) * std(
    csm_ep) / sqrt(length(csm));

end
```

10.4 State-space Model with One Binary, Two Continuous and a Spiking-Type Observation

10.4.1 Simulated Data Example

```
load('data_one_bin_two_cont_one_spk.mat');

delta = 0.005;

%% parameters

M = 5e5;    % maximum iterations
tol = 1e-5; % convergence criteria
```

```
b0 = zeros(1, M);   % binary GLM model
b1 = zeros(1, M);

r0 = zeros(1, M);   % continuous model
r1 = zeros(1, M);
vr = zeros(1, M);   % continuous model noise variance (1)

s0 = zeros(1, M);   % continuous model
s1 = zeros(1, M);
vs = zeros(1, M);   % continuous model noise variance (2)

ve = zeros(1, M);   % process noise variance
rho = zeros(1, M);  % random walk fogetting factor
alpha = zeros(1, M);    % input gain parameter

K = length(n);

x_pred = zeros(1, K);
v_pred = zeros(1, K);

x_updt = zeros(1, K);
v_updt = zeros(1, K);

x_smth = zeros(1, K);
v_smth = zeros(1, K);

p_updt = zeros(1, K);

A = zeros(1, K);
W = zeros(1, K);
CW = zeros(1, K);
C = zeros(1, K);

rpeaks = zeros(1, K * 50);
rpeaks(round(rpeak_locs / delta)) = 1;
rpeaks = reshape(rpeaks, [50, K])';

exception_counter = 0;

%% initial guesses

base_prob = sum(n) / length(n);
b0(1) = log(base_prob / (1 - base_prob));
b1(1) = 1;

r0(1) = 0.27154;
r1(1) = 0.5057;
vr(1) = 0.00187;

s0(1) = -0.73899;
s1(1) = 0.25324;
vs(1) = 0.00302;

ve(1) = 0.01883;
```

```
rho(1) = 0.99411;
alpha(1) = 0.00818;

theta = theta';

eta = -0.001;

%% main function

for m = 1:M

    for k = 1:K

        if (k == 1)
            x_pred(k) = x_smth(1);
            v_pred(k) = ve(m) + ve(m);
        else
            x_pred(k) = rho(m) * x_updt(k - 1) + alpha(m) * I(k);
            v_pred(k) = (rho(m) ^ 2) * v_updt(k - 1) + ve(m);
        end

        C(k) = v_pred(k) / (vr(m) * vs(m) + v_pred(k) * ((r1(m) ^
     2) * vs(m) + (s1(m) ^ 2) * vr(m)));

        try     % numerical issues can occur due to the integrals
            [temp1, temp2] = get_posterior_mode(x_pred(k), C(k),
    r(k), r0(m), r1(m), b0(m), b1(m), vr(m), n(k), s(k), s0(m),
    s1(m), vs(m), ...
                rpeaks(k, :), ul(k, :), delta, w(k, :, :), theta
    ', eta);
            x_updt(k) = temp1;

            p_updt(k) = 1 / (1 + exp((-1) * (b0(m) + b1(m) *
    x_updt(k))));
            v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(
    m)) + ((s1(m) ^ 2) / vs(m)) + (b1(m) ^ 2) * p_updt(k) * (1 -
    p_updt(k)) - temp2);
        catch
            x_updt(k) = x_pred(k);
            v_updt(k) = v_pred(k);
            exception_counter = exception_counter + 1;
        end

        if (mod(k, 100) == 0)
            fprintf('%d ', k);
        end

        if (mod(k, 2500) == 0)
            fprintf('\n');
        end
    end

    x_smth(K) = x_updt(K);
    v_smth(K) = v_updt(K);
```

```
    W(K) = v_smth(K) + (x_smth(K) ^ 2);

    A(1:(end - 1)) = rho(m) * v_updt(1:(end - 1)) ./ v_pred(2:end
    );

    for k = (K - 1):(-1):1
       x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
     1));
       v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
    v_pred(k + 1));

       CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
       W(k) = v_smth(k) + (x_smth(k) ^ 2);
    end

    if (m < M)

        R = get_linear_parameters(x_smth, W, r, K);
        S = get_linear_parameters(x_smth, W, s, K);

        Q = [sum(W(1:end - 1)) (I(2:end) * x_smth(1:(end - 1))');
     ...
             (I(2:end) * x_smth(1:(end - 1))') (I * I')] \ [sum(CW
    ); (I(2:end) * x_smth(2:end)')];

        b0(m + 1) = log(base_prob / (1 - base_prob));
        b1(m + 1) = 1;

        rho(m + 1) = Q(1, 1);

        if (Q(2, 1) < 0)    % in case this happens (generally
    only needed with experimental data)
            alpha(m + 1) = alpha(m);
        else
            alpha(m + 1) = Q(2, 1);
        end

        ve(m + 1) = (sum(W(2:end)) + (rho(m + 1) ^ 2) * sum(W(1:(
    end - 1))) - 2 * rho(m + 1) * sum(CW) - ...
            2 * alpha(m + 1) * (I(2:end) * x_smth(2:end)') + 2 *
    alpha(m + 1) * rho(m + 1) * (I(2:end) * x_smth(1:(end - 1))')
     + ...
            (alpha(m + 1) ^ 2) * (I * I')) / K;

        r0(m + 1) = R(1, 1);
        r1(m + 1) = R(2, 1);

        s0(m + 1) = S(1, 1);
        s1(m + 1) = S(2, 1);

        vr(m + 1) = get_maximum_variance(r, r0(m + 1), r1(m + 1),
     W, x_smth, K);
        vs(m + 1) = get_maximum_variance(s, s0(m + 1), s1(m + 1),
     W, x_smth, K);
```

```
        mean_dev = mean(abs([b0(m + 1) b1(m + 1) r0(m + 1) r1(m
    + 1) ve(m + 1) vr(m + 1) rho(m + 1) alpha(m + 1) s0(m + 1) s1
    (m + 1) vs(m + 1)] - ...
            [b0(m) b1(m) r0(m) r1(m) ve(m) vr(m) rho(m) alpha(m)
    s0(m) s1(m) vs(m)]));

        if mean_dev < tol
            fprintf('\n\nConverged at m = %d\n\n', m);
            break;
        else
            fprintf('m = %d\nb0 = %.18f\nb1 = %.18f\n\nr0 = %.18f
    \nr1 = %.18f\nvr = %.18f\n\ns0 = %.18f\ns1 = %.18f\nvs = %.18
    f\n\nve = %.18f\nrho = %.18f\nalpha = %.18f\n\ndev = %.18f\n\
    n', ...
            m + 1, b0(m + 1), b1(m + 1), r0(m + 1), r1(m + 1), vr
    (m + 1), s0(m + 1), s1(m + 1), vs(m + 1), ve(m + 1), rho(m +
    1), alpha(m + 1), mean_dev);

            x_pred = zeros(1, K);
            v_pred = zeros(1, K);

            x_updt = zeros(1, K);
            v_updt = zeros(1, K);

            x_smth(2:end) = zeros(1, K - 1);    % x_smth(1)
    needed for next iteration
            v_smth = zeros(1, K);

            p_updt = zeros(1, K);

            A = zeros(1, K);
            W = zeros(1, K);
            CW = zeros(1, K);
            C = zeros(1, K);
        end
    end
end

%% calculate confidence limits

p_smth = 1 ./ (1 + exp((-1) * (b0(m) + b1(m) * x_smth)));
r_smth = r0(m) + r1(m) * x_smth;
s_smth = s0(m) + s1(m) * x_smth;

lambda = zeros(K, 50);
mean_rr = zeros(K, 50);

for i = 1:K
    for j = 1:50
        w1 = [squeeze(w(i, j, :))' [eta x_smth(i)]];
        if (f(theta', ul(i, j), w1) > 1e-18)
            lambda(i, j) = fetch_lambda(theta', ul(i, j), w1);
        end
```

```
        mean_rr(i, j) =  mu(theta', w1);
    end
end

lambda_start_index = find(reshape(rpeaks', 1, numel(rpeaks)), 1);
lambda = reshape(lambda', 1, numel(lambda));

ll = get_log_likelihood(eta, rpeaks, ul, delta, w, theta', x_smth
    , v_smth);
ll_final = sum(nansum(ll));

%% plot graphs

figure;

mean_rr = reshape(mean_rr', 1, numel(mean_rr));
rri = diff(rpeak_locs);
rr_times = rpeak_locs(2:end);

index = (0:(K - 1));
fs_hyp = 4;
t_index = index / fs_hyp;
r_index = ((K - 1):(-1):0); % reverse index
transp = 0.3;

subplot(611);
hold on;
plot(t_index, p, 'b'); grid;
plot(t_index, p_smth, 'r');
plot((find(n == 0) - 1) / fs_hyp, 1.2 * max(p) * ones(length(find
    (n == 0))), 'ks', 'MarkerFaceColor', 'k', 'MarkerSize', 4);
plot((find(n == 1) - 1) / fs_hyp, 1.2 * max(p) * ones(length(find
    (n == 1))), 'gs', 'MarkerFaceColor', 'g', 'MarkerSize', 4);
ylabel('(a) p_{k}'); ylim([0 0.25]);
title('State Estimation with Simulated Data');

subplot(612);
hold on;
plot(t_index, r, 'b'); grid;
plot(t_index, r_smth, 'r');
ylabel('(b) r_{k}');

subplot(613);
hold on;
plot(t_index, s, 'b'); grid;
plot(t_index, s_smth, 'r');
ylabel('(c) s_{k}');

subplot(614);
hold on;
plot(t_index, x, 'b'); grid;
plot(t_index, x_smth, 'r');
plot((find(I == 0) - 1) / fs_hyp, (-8) * ones(length(find(I == 0)
    )), 'ks', 'MarkerFaceColor', 'k', 'MarkerSize', 4);
```

```
plot((find(I == 1) - 1) / fs_hyp, (-8) * ones(length(find(I == 1)
    )), 'cs', 'MarkerFaceColor', 'c', 'MarkerSize', 4);
ylabel('(d) x_{k}');

subplot(615);
hold on;
plot(rr_times, rri, 'o', 'Color', [1, 0.5, 0.25], '
    MarkerFaceColor', [1, 0.5, 0.25], 'MarkerSize', 2); grid;
mu_start_index = round(rpeak_locs(2) / delta);
plot(((0:(length(mean_rr(mu_start_index:end)) - 1)) * delta),
    mean_rr(mu_start_index:end), 'b');
ylabel('(e) rr_{i}'); xlim([0 t_index(end)]); xlabel('time (s)');

subplot(616);
qqplot(x_smth - x); grid;
title('QQ Plot - State Estimate', 'FontWeight', 'normal');
ylabel('(f) input quantiles');
xlabel('standard normal quantiles');

figure;
get_ks_plot(rpeak_locs, lambda(lambda_start_index:end), delta, 1)
    ;
ylabel({'Theoretical', 'Quantiles'}); xlabel('Empirical Quantiles
    ');
title('KS Plot');

%% supplementary functions

function [y, H2] = get_posterior_mode(x_pred, C, r, r0, r1, b0,
    b1, vr, n, s, s0, s1, vs, rpeaks, ul, delta, w_all, theta,
    eta)

    M = 200;    % maximum iterations

    it = zeros(1, M);
    func = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)

        H1 = zeros(1, 50);
        H2 = zeros(1, 50);

        for j = 1:50    % 5 ms -> 0.25 s (4 Hz for skin
    conductance)
            w = [squeeze(w_all(1, j, :))' [eta it(i)]];

            if (f(theta, ul(j), w) > 1e-18) %
                lambda = fetch_lambda(theta, ul(j), w);
                dl_dx = dlambda_dx(theta, ul(j), w);
```

```
                H1(j) = dl_dx * (rpeaks(j) - lambda * delta) /
    lambda;
                H2(j) = d2lambda_dx2(theta, ul(j), w) * (rpeaks(j
    ) - lambda * delta) / lambda - rpeaks(j) * (dl_dx ^ 2) / (
    lambda ^ 2);
            end
        end

        H1 = sum(H1);
        H2 = sum(H2);


        func(i) = it(i) - x_pred - C * (r1 * vs * (r - r0 - r1 *
    x_pred) + s1 * vr * (s - s0 - s1 * x_pred) + ...
            vr * vs * b1 * (n - (1 / (1 + exp((-1) * (b0 + b1 *
    it(i)))))) + vr * vs * H1);
        df(i) = 1 + C * vr * vs * ((b1 ^ 2) * exp(b0 + b1 * it(i)
    ) / ((1 + exp(b0 + b1 * it(i))) ^ 2) - H2);
        it(i + 1) = it(i) - func(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
           y = it(i + 1);
           return;
        end
    end

    error('Newton-Raphson failed to converge.');

end

function y = get_maximum_variance(z, r0, r1, W, x_smth, K)

    y = (z * z' + K * (r0 ^ 2) + (r1 ^ 2) * sum(W) ...
                - 2 * r0 * sum(z) - 2 * r1 * dot(x_smth, z) + 2 *
     r0 * r1 * sum(x_smth)) / K;

end

function y = get_linear_parameters(x_smth, W, z, K)

    y = [K sum(x_smth); sum(x_smth) sum(W)] \ [sum(z); sum(z .*
    x_smth)];

end

function [y] = f(theta, t, w)

    y = sqrt(theta(end) ./ (2 * pi * (t .^ 3))) .* ...
        exp((theta(end) * ((t - mu(theta, w)) .^ 2)) ./ ...
        ((-2) * (mu(theta, w) ^ 2) * t));

end

function [y] = intf(theta, t, w)
```

```
    y = integral(@(t)f(theta, t, w), 0, t);

end

function [y] = mu(theta, w)

    eta = w(end - 1);
    x = w(end);
    p = length(theta) - 2;

    y = theta(1) + theta(2:(2 + p - 1)) * w(1:p)' + eta * x;

end

function [y] = fetch_lambda(theta, t, w)

    cdf = intf(theta, t, w);
    y = f(theta, t, w) ./ (1 - cdf);

    if (cdf > 1)    % numerical issue
        y = 0;
    end

end

function [y] = df_dmu(theta, t, w)

    y = (theta(end) / (mu(theta, w) ^ 3)) * (f(theta, t, w) .* (t
     - mu(theta, w)));

end

function [y] = df_dx(theta, t, w)

    eta = w(end - 1);
    y = df_dmu(theta, t, w) .* eta;

end

function [y] = intdf_dx(theta, t, w)

    y = integral(@(t)df_dx(theta, t, w), 0, t);

end

function [y] = dlambda_dx(theta, t, w)

    cdf = intf(theta, t, w);

    if (cdf > 1)    % numerical issue
        y = 0;
    else
        y = ((1 - cdf) .* df_dx(theta, t, w) + ...
```

```
            f(theta, t, w) .* intdf_dx(theta, t, w)) ./ ((1 - cdf
    ) .^ 2);
    end
end

function [y] = d2f_dmu2(theta, t, w)

    y = theta(end) * (df_dmu(theta, t, w) .* ((t - mu(theta, w))
    / (mu(theta, w) ^ 3)) + ...
        f(theta, t, w) .* ((2 * mu(theta, w) - 3 * t) / (mu(theta
    , w) ^ 4)));

end

function [y] = d2f_dx2(theta, t, w)

    eta = w(end - 1);
    y = d2f_dmu2(theta, t, w) .* (eta ^ 2);

end

function [y] = intd2f_dx2(theta, t, w)

    y = integral(@(t)d2f_dx2(theta, t, w), 0, t);

end

function [y] = d2lambda_dx2(theta, t, w)

    y = (2 * dlambda_dx(theta, t, w) * (1 - intf(theta, t, w)) *
    intdf_dx(theta, t, w) + ...
         d2f_dx2(theta, t, w) * (1 - intf(theta, t, w)) + ...
         f(theta, t, w) * intd2f_dx2(theta, t, w)) / ((1 - intf(
    theta, t, w)) ^ 2);

end

function [y] = get_log_likelihood(eta, rpeaks, ul, delta, w_all,
    theta, x, v)

    K = length(x);
    y = zeros(K, 50);

    for k = 1:K
        for j = 1:50
            w = [squeeze(w_all(k, j, :))' [eta x(k)]];

            if (f(theta, ul(k, j), w) > 1e-18)

                lambda = fetch_lambda(theta, ul(k, j), w);
                dl_dx = dlambda_dx(theta, ul(k, j), w);
                d2l_dx2 = d2lambda_dx2(theta, ul(k, j), w);
                nkj = rpeaks(k, j);
```

```
                y(k, j) = nkj * log(delta * lambda) - delta *
    lambda + ...
                    (d2l_dx2 * (nkj - lambda * delta) / lambda -
    nkj * (dl_dx ^ 2) / (lambda ^ 2)) * v(k) * 0.5;
            end

        end
    end

end
```

10.4.2 Experimental Data Example

```
load('expm_data_one_bin_two_cont_one_spk.mat');

delta = 0.005;
min_scr_thresh = 0.015;
min_scr_prom = min_scr_thresh;
fs = 4;
epoch = 10;

stim = s_data.aug_stim;
ph = s_data.ph;
tn = s_data.tn;
rpeaks = s_data.rpeaks;
ul = s_data.ul;

[pks, locs] = findpeaks(ph, 'MinPeakHeight', min_scr_thresh, '
    MinPeakProminence', min_scr_prom);
r = interp1([1 locs length(ph)], log([ph(find(ph > 0, 1)) pks ph(
    end)]), 1:length(s_data.ph), 'cubic');

s = tn;
n = zeros(1, length(r));
I = zeros(1, length(r));

n(locs) = 1;
I(stim) = 1;

std_s = std(s);
std_r = std(r);

s = s / std_s;
r = r / std_r;

%% parameters

M = 5e5;    % maximum iterations
tol = 1e-6; % convergence criteria

b0 = zeros(1, M);   % binary GLM model
b1 = zeros(1, M);
```

```
r0 = zeros(1, M);   % continuous GLM model
r1 = zeros(1, M);
vr = zeros(1, M);   % continuous GLM model noise variance (1)

s0 = zeros(1, M);   % continuous GLM
s1 = zeros(1, M);
vs = zeros(1, M);   % continuous GLM model noise variance (2)

ve = zeros(1, M);   % process noise variance
rho = zeros(1, M);  % random walk correlation
alpha = zeros(1, M);    % input gain parameter

K = length(n);

x_pred = zeros(1, K);
v_pred = zeros(1, K);

x_updt = zeros(1, K);
v_updt = zeros(1, K);

x_smth = zeros(1, K);
v_smth = zeros(1, K);

p_updt = zeros(1, K);

A = zeros(1, K);
W = zeros(1, K);
CW = zeros(1, K);
C = zeros(1, K);

exception_counter = 0;

%% initial guesses

base_prob = sum(n) / length(n);
b0(1) = log(base_prob / (1 - base_prob));
b1(1) = 1;
r0(1) = r(1); % guess it's the first value of r
r1(1)  = 1;
s0(1) = s(1);
s1(1) = 1;
vr(1) = 0.05;
vs(1) = 0.05;
ve(1) = 0.05;
rho(1) = 1;
alpha(1) = 0.5;

theta = s_data.theta;

eta = -0.001;

%% main function
```

```
for m = 1:M

    for k = 1:K

        if (k == 1)
            x_pred(k) = x_smth(1);
            v_pred(k) = ve(m) + ve(m);
        else
            x_pred(k) = rho(m) * x_updt(k - 1) + alpha(m) * I(k);
            v_pred(k) = (rho(m) ^ 2) * v_updt(k - 1) + ve(m);
        end

        C(k) = v_pred(k) / (vr(m) * vs(m) + v_pred(k) * ((r1(m) ^
     2) * vs(m) + (s1(m) ^ 2) * vr(m)));

        try     % numerical issues can occur due to the integrals
            [temp1, temp2] = get_posterior_mode(x_pred(k), C(k),
    r(k), r0(m), r1(m), b0(m), b1(m), vr(m), n(k), s(k), s0(m),
    s1(m), vs(m), ...
                rpeaks(k, :), ul(k, :), delta, s_data.w(k, :, :),
     'theta', eta);
            x_updt(k) = temp1;

            p_updt(k) = 1 / (1 + exp((-1) * (b0(m) + b1(m) *
    x_updt(k))));
            v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(
    m)) + ((s1(m) ^ 2) / vs(m)) + (b1(m) ^ 2) * p_updt(k) * (1 -
    p_updt(k)) - temp2);
        catch
            x_updt(k) = x_pred(k);
            v_updt(k) = v_pred(k);
            exception_counter = exception_counter + 1;
        end

        if (mod(k, 100) == 0)
            fprintf('%d ', k);
        end

        if (mod(k, 2500) == 0)
            fprintf('\n');
        end
    end

    x_smth(K) = x_updt(K);
    v_smth(K) = v_updt(K);
    W(K) = v_smth(K) + (x_smth(K) ^ 2);

    A(1:(end - 1)) = rho(m) * v_updt(1:(end - 1)) ./ v_pred(2:end
    );

    for k = (K - 1):(-1):1
       x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
     1));
```

```
    v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
v_pred(k + 1));

    CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
    W(k) = v_smth(k) + (x_smth(k) ^ 2);
end

if (m < M)

    R = get_linear_parameters(x_smth, W, r, K);
    S = get_linear_parameters(x_smth, W, s, K);

    Q = [sum(W(1:end - 1)) (I(2:end) * x_smth(1:(end - 1))');
 ...
        (I(2:end) * x_smth(1:(end - 1))') (I * I')] \ [sum(CW
); (I(2:end) * x_smth(2:end)')];

    bb = fsolve(@(b) binary_parameter_derivatives(b, n,
x_smth, v_smth), [-5 1], optimset('Display','off'));
    b0(m + 1) = bb(1);
    b1(m + 1) = bb(2);

    rho(m + 1) = Q(1, 1);

    if (Q(2, 1) < 0)    % check in case this happens
        alpha(m + 1) = alpha(m);
    else
        alpha(m + 1) = Q(2, 1);
    end

    ve(m + 1) = (sum(W(2:end)) + (rho(m + 1) ^ 2) * sum(W(1:(
end - 1))) - 2 * rho(m + 1) * sum(CW) - ...
        2 * alpha(m + 1) * (I(2:end) * x_smth(2:end)') + 2 *
alpha(m + 1) * rho(m + 1) * (I(2:end) * x_smth(1:(end - 1))')
 + ...
        (alpha(m + 1) ^ 2) * (I * I')) / K;

    if (abs(get_maximum_variance(r, R(1, 1), R(2, 1), W,
x_smth, K) - get_maximum_variance(s, S(1, 1), S(2, 1), W,
x_smth, K)) > 0.1)   % terminate once overfitting is detected
        break;
    else
        r0(m + 1) = R(1, 1);
        r1(m + 1) = R(2, 1);

        s0(m + 1) = S(1, 1);
        s1(m + 1) = S(2, 1);

        vr(m + 1) = get_maximum_variance(r, r0(m + 1), r1(m +
 1), W, x_smth, K);
        vs(m + 1) = get_maximum_variance(s, s0(m + 1), s1(m +
 1), W, x_smth, K);

    end
```

```
        mean_dev = mean(abs([b0(m + 1) b1(m + 1) r0(m + 1) r1(m
    + 1) ve(m + 1) vr(m + 1) rho(m + 1) alpha(m + 1) s0(m + 1) s1
    (m + 1) vs(m + 1)] - ...
            [b0(m) b1(m) r0(m) r1(m) ve(m) vr(m) rho(m) alpha(m)
    s0(m) s1(m) vs(m)]));

        if mean_dev < tol
            fprintf('Converged at m = %d\n\n', m);
            break;
        else
            fprintf('m = %d\nb0 = %.18f\nb1 = %.18f\n\nr0 = %.18f
    \nr1 = %.18f\nvr = %.18f\n\ns0 = %.18f\ns1 = %.18f\nvs = %.18
    f\n\nve = %.18f\nrho = %.18f\nalpha = %.18f\n\ndev = %.18f\n\
    n', ...
            m + 1, b0(m + 1), b1(m + 1), r0(m + 1), r1(m + 1), vr
    (m + 1), s0(m + 1), s1(m + 1), vs(m + 1), ve(m + 1), rho(m +
    1), alpha(m + 1), mean_dev);

            x_pred = zeros(1, K);
            v_pred = zeros(1, K);

            x_updt = zeros(1, K);
            v_updt = zeros(1, K);

            x_smth(2:end) = zeros(1, K - 1);    % x_smth(1)
    needed for next iteration
            v_smth = zeros(1, K);

            p_updt = zeros(1, K);

            A = zeros(1, K);
            W = zeros(1, K);
            CW = zeros(1, K);
            C = zeros(1, K);
        end
    end
end

%% calculate confidence limits

p_smth = 1 ./ (1 + exp((-1) * (b0(m) + b1(m) * x_smth)));
lcl_fp = zeros(1, K);
ucl_fp = zeros(1, K);

r_smth = exp((r0(m) + r1(m) * x_smth) * std_r);
s_smth = (s0(m) + s1(m) * x_smth) * std_s;

skn_avg = get_trial_averages(s_data, x_smth, epoch, fs, 'skn');
x_avg = get_trial_averages(s_data, x_smth, epoch, fs, 'x_smth');

t_epoch = ((-1):(1 / fs):(epoch - 1 - (1 / fs)));
tr_epoch = ((epoch - 1 - (1 / fs)):(-1 / fs):(-1));
```

```
fprintf('Plotting\n');

lambda = zeros(K, 50);
mean_rr = zeros(K, 50);

for i = 1:K
    for j = 1:50
        w = [squeeze(s_data.w(i, j, :))' [eta x_smth(i)]];
        if (f(theta', ul(i, j), w) > 1e-18)
            lambda(i, j) = fetch_lambda(theta', ul(i, j), w);
        end
        mean_rr(i, j) =  mu(theta', w);
    end
end

lambda_start_index = find(reshape(s_data.rpeaks', 1, numel(s_data
    .rpeaks)), 1);
lambda = reshape(lambda', 1, numel(lambda));

ll = get_log_likelihood(eta, rpeaks, ul, delta, s_data.w, theta',
     x_smth, v_smth);
ll_final = sum(nansum(ll));

%% plot graphs

mean_rr = reshape(mean_rr', 1, numel(mean_rr));
rri = diff(s_data.rpeak_locs);
rr_times = s_data.rpeak_locs(2:end);

index = (0:(K - 1));
t_index = index / fs;
r_index = ((K - 1):(-1):0);
transp = 0.3;

subplot(711);
plot(t_index, s_data.x, 'color', [(102 / 255) 0 (204 / 255)]);
ylabel('(a) z_{k}'); grid; xlim([0 t_index(end)]);
set(gca,'xticklabel', []);
ylim([(min(s_data.x) - 0.1) (max(s_data.x) + 0.1)]);
title('State Estimation with Experimental Data');

subplot(712);
hold on;
plot(find(n == 0) / fs, max(p_smth) * 1.3 * ones(length(find(n ==
     0))), 'ks', 'MarkerFaceColor', 'k', 'MarkerSize', 3);
plot(find(n == 1) / fs, max(p_smth) * 1.3 * ones(length(find(n ==
     1))), 'gs', 'MarkerFaceColor', 'g', 'MarkerSize', 3);
plot(t_index, p_smth, 'r');

ylabel('(b) p_{k}');
xlim([0 t_index(end)]); ylim([0 (max(p_smth) * 1.5)]); grid;
set(gca,'xticklabel', []);

subplot(713);
```

```
hold on;
plot(t_index, r_smth, ':', 'color', [0 0.3 0], 'linewidth', 1.5);
plot(t_index, exp(r * std_r), 'color', [0 0.9 0]);

ylabel('(c) e^{r_{k}}'); grid;
xlim([0 t_index(end)]);
set(gca,'xticklabel', []);

subplot(714);
hold on;
plot(t_index, s_smth, ':', 'color', [0.5 (25 / 255) (66 / 255)],
    'linewidth', 1.5);
plot(t_index, s * std_s, 'color', [1 0.5 (179 / 255)]);

ylabel('(d) s_{k}');
xlim([0 t_index(end)]); grid;
set(gca,'xticklabel', []);

subplot(715);
hold on;
plot(t_index, x_smth, 'color', 'b');
plot(find(I == 0) / fs, (min(x_smth) - 0.5) * ones(length(find(I
    == 0))), 'ks', 'MarkerFaceColor', 'k', 'MarkerSize', 3);
plot(find(I == 1) / fs, (min(x_smth) - 0.5) * ones(length(find(I
    == 1))), 'cs', 'MarkerFaceColor', 'c', 'MarkerSize', 3);

ylabel('(e) x_{k}'); ylim([(min(x_smth) - 1) (max(x_smth) + 1)]);
xlim([0 t_index(end)]); grid; set(gca,'xticklabel', []);

subplot(716);
hold on;
plot(rr_times / 60, rri, 'o', 'Color', [1, 0.5, 0.25], '
    MarkerFaceColor', [1, 0.5, 0.25], 'MarkerSize', 3); grid;
mu_start_index = round(s_data.rpeak_locs(2) / delta);
plot(((0:(length(mean_rr(mu_start_index:end)) - 1)) * delta) /
    60, mean_rr(mu_start_index:end), 'b');
ylabel('(f) rr_{i}'); xlim([0 t_index(end)] / 60); xlabel('time (
    min)');

subplot(7, 2, 13);
hold on;

plot(t_epoch, skn_avg(7, :), 'color', [0 0.8 0], 'linewidth',
    1.5);
fill([t_epoch, tr_epoch], [skn_avg(8, :) fliplr(skn_avg(9, :))],
    'g', 'EdgeColor', 'none', 'FaceAlpha', 0.2);

plot(t_epoch, skn_avg(4, :), 'm', 'linewidth', 1.5);
fill([t_epoch, tr_epoch], [skn_avg(5, :) fliplr(skn_avg(6, :))],
    'm', 'EdgeColor', 'none', 'FaceAlpha', 0.2);

plot(t_epoch, skn_avg(1, :), 'r', 'linewidth', 1.5);
fill([t_epoch, tr_epoch], [skn_avg(2, :) fliplr(skn_avg(3, :))],
    'r', 'EdgeColor', 'none', 'FaceAlpha', 0.2);
```

```
xlim([t_epoch(1) t_epoch(end)]);

grid;
xlabel('time (s)'); ylabel('(g) z_{k}');

subplot(7, 2, 14);
hold on;

plot(t_epoch, x_avg(7, :), 'color', [0 0.8 0], 'linewidth', 1.5);
fill([t_epoch, tr_epoch], [x_avg(8, :) fliplr(x_avg(9, :))], 'g',
     'EdgeColor', 'none', 'FaceAlpha', 0.2);

plot(t_epoch, x_avg(4, :), 'm', 'linewidth', 1.5);
fill([t_epoch, tr_epoch], [x_avg(5, :) fliplr(x_avg(6, :))], 'm',
     'EdgeColor', 'none', 'FaceAlpha', 0.2);

plot(t_epoch, x_avg(1, :), 'r', 'linewidth', 1.5);
fill([t_epoch, tr_epoch], [x_avg(2, :) fliplr(x_avg(3, :))], 'r',
     'EdgeColor', 'none', 'FaceAlpha', 0.2);

xlim([t_epoch(1) t_epoch(end)]);

grid;
xlabel('time (s)'); ylabel('(h) x_{k}');

figure;
get_ks_plot(s_data.rpeak_locs, lambda(lambda_start_index:end),
    delta, 1);

%% supplementary functions

function [y, H2] = get_posterior_mode(x_pred, C, r, r0, r1, b0,
    b1, vr, n, s, s0, s1, vs, rpeaks, ul, delta, w_all, theta,
    eta)

    M = 200;    % maximum iterations

    it = zeros(1, M);
    func = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)

        H1 = zeros(1, 50);
        H2 = zeros(1, 50);

        for j = 1:50    % 5 ms -> 0.25 s (4 Hz for skin
    conductance)
            w = [squeeze(w_all(1, j, :))' [eta it(i)]];

            if (f(theta, ul(j), w) > 1e-18) %
```

```
                lambda = fetch_lambda(theta, ul(j), w);
                dl_dx = dlambda_dx(theta, ul(j), w);

                H1(j) = dl_dx * (rpeaks(j) - lambda * delta) /
    lambda;
                H2(j) = d2lambda_dx2(theta, ul(j), w) * (rpeaks(j
    ) - lambda * delta) / lambda - rpeaks(j) * (dl_dx ^ 2) / (
    lambda ^ 2);
            end
        end

        H1 = sum(H1);
        H2 = sum(H2);


        func(i) = it(i) - x_pred - C * (r1 * vs * (r - r0 - r1 *
    x_pred) + s1 * vr * (s - s0 - s1 * x_pred) + ...
            vr * vs * b1 * (n - (1 / (1 + exp((-1) * (b0 + b1 *
    it(i)))))) + vr * vs * H1);
        df(i) = 1 + C * vr * vs * ((b1 ^ 2) * exp(b0 + b1 * it(i)
    ) / ((1 + exp(b0 + b1 * it(i))) ^ 2) - H2);
        it(i + 1) = it(i) - func(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
           y = it(i + 1);
           return;
        end
    end

    error('Newton-Raphson failed to converge.');

end

function y = binary_parameter_derivatives(b, n, x_smth, v_smth)

    y = zeros(1, 2);
    K = length(n);

    b0 = b(1);
    b1 = b(2);
    p = zeros(1, K);

    for k = 1:K
        p(k) = 1 / (1 + exp((-1) * (b0 + b1 * x_smth(k))));
        y(1) = y(1) + n(k) - p(k) - 0.5 * v_smth(k) * (b1 ^ 2) *
    p(k) * (1 - p(k)) * (1 - 2 * p(k));
        y(2) = y(2) + n(k) * x_smth(k) - x_smth(k) * p(k) - 0.5 *
     v_smth(k) * b1 * p(k) * (1 - p(k)) * (2 + x_smth(k) * b1 *
    (1 - 2 * p(k)));
    end

end

function y = get_maximum_variance(z, r0, r1, W, x_smth, K)
```

```
    y = (z * z' + K * (r0 ^ 2) + (r1 ^ 2) * sum(W) ...
                - 2 * r0 * sum(z) - 2 * r1 * dot(x_smth, z) + 2 *
     r0 * r1 * sum(x_smth)) / K;

end

function y = get_linear_parameters(x_smth, W, z, K)

    y = [K sum(x_smth); sum(x_smth) sum(W)] \ [sum(z); sum(z .*
    x_smth)];

end

function y = get_trial_averages(s, x_smth, epoch, fs, option)

    y = zeros(9, epoch * fs);

    csm_ep = zeros(length(s.csm), epoch * fs);
    csp_us_ep = zeros(length(s.csp_us), epoch * fs);
    csp_nus_ep = zeros(length(s.csp_nus), epoch * fs);

    csm = s.csm;
    csp_us = s.csp_us;
    csp_nus = s.csp_nus;

    if strcmp(option, 'x_smth')

        for j = 1:length(csm)
            csm_ep(j, :) = x_smth((s.stim(csm(j)) - fs ):(s.stim(
    csm(j)) + 9 * fs - 1));
        end

        for j = 1:length(csp_us)
            csp_us_ep(j, :) = x_smth((s.stim(csp_us(j)) - fs):(s.
    stim(csp_us(j)) + 9 * fs - 1));
        end

        for j = 1:length(csp_nus)
            csp_nus_ep(j, :) = x_smth((s.stim(csp_nus(j)) - fs):(
    s.stim(csp_nus(j)) + 9 * fs - 1));
        end

    elseif strcmp(option, 'skn')

        for j = 1:length(csm)
            csm_ep(j, :) = s.x((s.stim(csm(j)) - fs ):(s.stim(csm
    (j)) + 9 * fs - 1));
        end

        for j = 1:length(csp_us)
            csp_us_ep(j, :) = s.x((s.stim(csp_us(j)) - fs):(s.
    stim(csp_us(j)) + 9 * fs - 1));
        end
```

```
        for j = 1:length(csp_nus)
            csp_nus_ep(j, :) = s.x((s.stim(csp_nus(j)) - fs):(s.
    stim(csp_nus(j)) + 9 * fs - 1));
        end

    end

    y(1, :) = mean(csp_us_ep);
    y(2, :) = mean(csp_us_ep) + tinv(0.975, length(csp_us) - 1) *
     std(csp_us_ep) / sqrt(length(csp_us));
    y(3, :) = mean(csp_us_ep) + tinv(0.025, length(csp_us) - 1) *
     std(csp_us_ep) / sqrt(length(csp_us));

    y(4, :) = mean(csp_nus_ep);
    y(5, :) = mean(csp_nus_ep) + tinv(0.975, length(csp_nus) - 1)
     * std(csp_nus_ep) / sqrt(length(csp_nus));
    y(6, :) = mean(csp_nus_ep) + tinv(0.025, length(csp_nus) - 1)
     * std(csp_nus_ep) / sqrt(length(csp_nus));

    y(7, :) = mean(csm_ep);
    y(8, :) = mean(csm_ep) + tinv(0.975, length(csm) - 1) * std(
    csm_ep) / sqrt(length(csm));
    y(9, :) = mean(csm_ep) + tinv(0.025, length(csm) - 1) * std(
    csm_ep) / sqrt(length(csm));

end

function [y] = f(theta, t, w)

    y = sqrt(theta(end) ./ (2 * pi * (t .^ 3))) .* ...
        exp((theta(end) * ((t - mu(theta, w)) .^ 2)) ./ ...
        ((-2) * (mu(theta, w) ^ 2) * t));

end

function [y] = intf(theta, t, w)

    y = integral(@(t)f(theta, t, w), 0, t);

end

function [y] = mu(theta, w)

    eta = w(end - 1);
    x = w(end);
    p = length(theta) - 2;

    y = theta(1) + theta(2:(2 + p - 1)) * w(1:p)' + eta * x;

end

function [y] = fetch_lambda(theta, t, w)
```

```
    cdf = intf(theta, t, w);
    y = f(theta, t, w) ./ (1 - cdf);

    if (cdf > 1)    % numerical issue
        y = 0;
    end

end

function [y] = df_dmu(theta, t, w)

    y = (theta(end) / (mu(theta, w) ^ 3)) * (f(theta, t, w) .* (t
     - mu(theta, w)));

end

function [y] = df_dx(theta, t, w)

    eta = w(end - 1);
    y = df_dmu(theta, t, w) .* eta;

end

function [y] = intdf_dx(theta, t, w)

    y = integral(@(t)df_dx(theta, t, w), 0, t);

end

function [y] = dlambda_dx(theta, t, w)

    cdf = intf(theta, t, w);

    if (cdf > 1)    % numerical issue
        y = 0;
    else
        y = ((1 - cdf) .* df_dx(theta, t, w) + ...
            f(theta, t, w) .* intdf_dx(theta, t, w)) ./ ((1 - cdf
    ) .^ 2);
    end
end

function [y] = d2f_dmu2(theta, t, w)

    y = theta(end) * (df_dmu(theta, t, w) .* ((t - mu(theta, w))
    / (mu(theta, w) ^ 3)) + ...
        f(theta, t, w) .* ((2 * mu(theta, w) - 3 * t) / (mu(theta
    , w) ^ 4)));

end

function [y] = d2f_dx2(theta, t, w)

    eta = w(end - 1);
```

```
    y = d2f_dmu2(theta, t, w) .* (eta ^ 2);

end

function [y] = intd2f_dx2(theta, t, w)

    y = integral(@(t)d2f_dx2(theta, t, w), 0, t);

end

function [y] = d2lambda_dx2(theta, t, w)

    y = (2 * dlambda_dx(theta, t, w) * (1 - intf(theta, t, w)) *
    intdf_dx(theta, t, w) + ...
         d2f_dx2(theta, t, w) * (1 - intf(theta, t, w)) + ...
         f(theta, t, w) * intd2f_dx2(theta, t, w)) / ((1 - intf(
    theta, t, w)) ^ 2);

end

function [y] = get_log_likelihood(eta, rpeaks, ul, delta, w_all,
    theta, x, v)

    K = length(x);
    y = zeros(K, 50);

    for k = 1:K
        for j = 1:50
            w = [squeeze(w_all(k, j, :))' [eta x(k)]];

            if (f(theta, ul(k, j), w) > 1e-18)

                lambda = fetch_lambda(theta, ul(k, j), w);
                dl_dx = dlambda_dx(theta, ul(k, j), w);
                d2l_dx2 = d2lambda_dx2(theta, ul(k, j), w);
                nkj = rpeaks(k, j);

                y(k, j) = nkj * log(delta * lambda) - delta *
    lambda + ...
                    (d2l_dx2 * (nkj - lambda * delta) / lambda -
    nkj * (dl_dx ^ 2) / (lambda ^ 2)) * v(k) * 0.5;
            end

        end
    end

end
```

10.5 State-space Model with One MPP Observation

10.5.1 Simulated Data Example

```
load('data_one_mpp.mat');

K = length(n);

pt = find(n > 0);

M = 5e4;
ve = zeros(1, M);   % process noise variance
r0 = zeros(1, M);   % linear model coefficients (continuous
    variable)
r1 = zeros(1, M);   % linear model coefficients (continuous
    variable)
vr = zeros(1, M);   % sensor noise variance (continuous variable)

x_pred = zeros(1, K);
v_pred = zeros(1, K);

x_updt = zeros(1, K);
v_updt = zeros(1, K);

x_smth = zeros(1, K);
v_smth = zeros(1, K);

p_updt = zeros(1, K);

base_prob = sum(n) / length(n);
tol = 1e-8; % convergence criteria

A = zeros(1, K);
W = zeros(1, K);
CW = zeros(1, K);
C = zeros(1, K);

ve(1) = 0.005;
x_smth(1) = 0;
r0(1) = 0.003;
r1(1) = 0.001;
vr(1) = 0.002;
b0 = log(base_prob / (1 - base_prob));

for m = 1:M

    for k = 1:K

        if (k == 1)
            x_pred(k) = x_smth(1);
            v_pred(k) = ve(m) + ve(m);
        else
```

```
            x_pred(k) = x_updt(k - 1);
            v_pred(k) = v_updt(k - 1) + ve(m);
        end
        x_updt(k) = get_posterior_mode(x_pred(k), v_pred(k), r(k)
    , r0(m), r1(m), vr(m), b0, n(k));
        p_updt(k) = 1 / (1 + exp((-1) * (b0 + x_updt(k))));

        if (n(k) == 0)
            v_updt(k) = 1 / ((1 / v_pred(k)) + p_updt(k) * (1 -
    p_updt(k)));
        elseif (n(k) == 1)
            v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(
    m)) + p_updt(k) * (1 - p_updt(k)));
        end
    end

    x_smth(K) = x_updt(K);
    v_smth(K) = v_updt(K);
    W(K) = v_smth(K) + (x_smth(K) ^ 2);

    A(1:(end - 1)) = v_updt(1:(end - 1)) ./ v_pred(2:end);

    for k = (K - 1):(-1):1
       x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
     1));
       v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
    v_pred(k + 1));

       CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
       W(k) = v_smth(k) + (x_smth(k) ^ 2);
    end

    if (m < M)

        R = get_linear_parameters(x_smth, W, r, pt);

        r0(m + 1) = R(1, 1);
        r1(m + 1) = R(2, 1);
        vr(m + 1) = get_maximum_variance(r, r0(m + 1), r1(m + 1),
     W, x_smth, pt);

        ve(m + 1) = (sum(W(2:end)) + sum(W(1:(end - 1))) - 2 *
    sum(CW)) / K;

        mean_dev = mean(abs([ve(m + 1) r0(m + 1) r1(m + 1) vr(m +
     1)] - [ve(m) r0(m) r1(m) vr(m)]));

        if mean_dev < tol
            fprintf('m = %d\nx0 = %.18f\n\nr0 = %.18f\nr1 = %.18f
    \nvr = %.18f\nve = %.18f\n\n', m, x_smth(1), r0(m), r1(m), vr
    (m), ve(m));
            fprintf('Converged at m = %d\n\n', m);
            break;
        else
```

```
            fprintf('m = %d\nx0 = %.18f\n\nr0 = %.18f\nr1 = %.18f
    \nvr = %.18f\nve = %.18f\n\n', m, x_smth(1), r0(m + 1), r1(m
    + 1), vr(m + 1), ve(m + 1));

            x_pred = zeros(1, K);
            v_pred = zeros(1, K);

            x_updt = zeros(1, K);
            v_updt = zeros(1, K);

            x_smth(2:end) = zeros(1, K - 1);    % x_smth(1)
    needed for next iteration
            v_smth = zeros(1, K);

            p_updt = zeros(1, K);

            A = zeros(1, K);
            W = zeros(1, K);
            CW = zeros(1, K);
            C = zeros(1, K);
        end
    end
end

p_updt = 1 ./ (1 + exp((-1) * (b0 + x_updt)));
p_smth = 1 ./ (1 + exp((-1) * (b0 + x_smth)));
r_smth = r0(m) + r1(m) * x_smth;

r_plot = NaN * ones(1, K);
r_plot(pt) = r(pt);

figure;
subplot(411);
hold on;
stem(r_plot, 'fill', 'color', 'b', 'markersize', 4);
plot(r_smth, 'r-.', 'linewidth', 1.5);
ylabel('(a) n_{k}, r_{k}');
title('Estimation with Simulated Data');
grid;

subplot(412);
hold on;
plot(p, 'b');
plot(p_smth, 'r-.', 'linewidth', 1.5);
ylabel('(b) p_{k}');
grid;

subplot(413);
hold on;
plot(x, 'b');
plot(x_smth, 'r-.', 'linewidth', 1.5);
ylabel('(c) x_{k}');
xlabel('time index');
grid;
```

```
subplot(414);
qqplot(x - x_smth);
title('QQ Plot - State Estimate', 'FontWeight', 'Normal');
ylabel('(d) input quantiles');
xlabel('standard normal quantiles');
grid;

function [y] = get_posterior_mode(x_pred, v_pred, z, r0, r1, vr,
    b0, n)

    M = 100;    % maximum iterations
    y = NaN;

    it = zeros(1, M);
    f = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)
        if (n == 0)
            C = v_pred;
            f(i) = it(i) - x_pred - C * (n - exp(b0 + it(i)) / (1
     + exp(b0 + it(i))));
            df(i) = 1 + C * exp(b0 + it(i)) / (1 + exp(b0 + it(i)
    )) ^ 2;
        elseif (n == 1)
            C = v_pred / ((r1 ^ 2) * v_pred + vr);
            f(i) = it(i) - x_pred - C * (r1 * (z - r0 - r1 *
    x_pred) + vr * (n - (1 / (1 + exp((-1) * (b0 + it(i)))))));
            df(i) = 1 + C * vr * exp(b0 + it(i)) / ((1 + exp(b0 +
     it(i))) ^ 2);
        end

        it(i + 1)  = it(i) - f(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
            y = it(i + 1);
            return
        end
    end

    error('Newton-Raphson failed to converge.');

end

function y = get_maximum_variance(z, r0, r1, W, x_smth, pt)

    x_smth = x_smth(pt);
    W = W(pt);
    z = z(pt);
    K = length(pt);
```

```
    y = (z * z' + K * (r0 ^ 2) + (r1 ^ 2) * sum(W) ...
                - 2 * r0 * sum(z) - 2 * r1 * dot(x_smth, z) + 2 *
     r0 * r1 * sum(x_smth)) / K;
end

function y = get_linear_parameters(x_smth, W, z, pt)

    x_smth = x_smth(pt);
    W = W(pt);
    z = z(pt);
    K = length(pt);

    y = [K sum(x_smth); sum(x_smth) sum(W)] \ [sum(z); sum(z .*
    x_smth)];

end
```

10.5.2 Experimental Data Example

```
load('expm_data_one_mpp.mat');

K = length(u);
n = zeros(1, K);

pt = find(u > 0);
n(pt) = 1;
r = u;

M = 5e4;
ve = zeros(1, M);   % process noise variance
r0 = zeros(1, M);   % linear model coefficients (continuous
    variable)
r1 = zeros(1, M);   % linear model coefficients (continuous
    variable)
vr = zeros(1, M);   % sensor noise variance (continuous variable)

x_pred = zeros(1, K);
v_pred = zeros(1, K);

x_updt = zeros(1, K);
v_updt = zeros(1, K);

x_smth = zeros(1, K);
v_smth = zeros(1, K);

p_updt = zeros(1, K);

base_prob = sum(n) / length(n);
tol = 1e-8; % convergence criteria

A = zeros(1, K);
W = zeros(1, K);
```

```
CW = zeros(1, K);
C = zeros(1, K);

ve(1) = 0.005;
x_smth(1) = 0;
r0(1) = 0.003;
r1(1) = 0.001;
vr(1) = 0.002;
b0 = log(base_prob / (1 - base_prob));

for m = 1:M

    for k = 1:K

        if (k == 1)
            x_pred(k) = x_smth(1);
            v_pred(k) = ve(m) + ve(m);
        else
            x_pred(k) = x_updt(k - 1);
            v_pred(k) = v_updt(k - 1) + ve(m);
        end
        x_updt(k) = get_posterior_mode(x_pred(k), v_pred(k), r(k)
    , r0(m), r1(m), vr(m), b0, n(k));
        p_updt(k) = 1 / (1 + exp((-1) * (b0 + x_updt(k))));

        if (n(k) == 0)
            v_updt(k) = 1 / ((1 / v_pred(k)) + p_updt(k) * (1 -
    p_updt(k)));
        elseif (n(k) == 1)
            v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(
    m)) + p_updt(k) * (1 - p_updt(k)));
        end
    end

    x_smth(K) = x_updt(K);
    v_smth(K) = v_updt(K);
    W(K) = v_smth(K) + (x_smth(K) ^ 2);

    A(1:(end - 1)) = v_updt(1:(end - 1)) ./ v_pred(2:end);

    for k = (K - 1):(-1):1
       x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
     1));
       v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
    v_pred(k + 1));

       CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
       W(k) = v_smth(k) + (x_smth(k) ^ 2);
    end

    if (m < M)

        R = get_linear_parameters(x_smth, W, r, pt);
```

```
        if R(2, 1) > 0
            r0(m + 1) = R(1, 1);
            r1(m + 1) = R(2, 1);
            vr(m + 1) = get_maximum_variance(r, r0(m + 1), r1(m +
     1), W, x_smth, pt);
        else % a check with experimental data (in case this
    happens)
            fprintf('m = %d\nx0 = %.18f\n\nr0 = %.18f\nr1 = %.18f
    \nvr = %.18f\nve = %.18f\n\n', m, x_smth(1), r0(m), r1(m), vr
    (m), ve(m));
            fprintf('Converged at m = %d\n\n', m);
            break;
        end

        ve(m + 1) = (sum(W(2:end)) + sum(W(1:(end - 1))) - 2 *
    sum(CW)) / K;

        mean_dev = mean(abs([ve(m + 1) r0(m + 1) r1(m + 1) vr(m +
     1)] - [ve(m) r0(m) r1(m) vr(m)]));

        if mean_dev < tol
            fprintf('m = %d\nx0 = %.18f\n\nr0 = %.18f\nr1 = %.18f
    \nvr = %.18f\nve = %.18f\n\n', m, x_smth(1), r0(m), r1(m), vr
    (m), ve(m));
            fprintf('Converged at m = %d\n\n', m);
            break;
        else
            fprintf('m = %d\nx0 = %.18f\n\nr0 = %.18f\nr1 = %.18f
    \nvr = %.18f\nve = %.18f\n\n', m, x_smth(1), r0(m + 1), r1(m
    + 1), vr(m + 1), ve(m + 1));

            x_pred = zeros(1, K);
            v_pred = zeros(1, K);

            x_updt = zeros(1, K);
            v_updt = zeros(1, K);

            x_smth(2:end) = zeros(1, K - 1);    % x_smth(1)
    needed for next iteration
            v_smth = zeros(1, K);

            p_updt = zeros(1, K);

            A = zeros(1, K);
            W = zeros(1, K);
            CW = zeros(1, K);
            C = zeros(1, K);
        end
    end
end

p_smth = 1 ./ (1 + exp((-1) * (b0 + x_smth)));
r_smth = r0(m) + r1(m) * x_smth;
```

```
lcl_x = norminv(0.025, x_smth, sqrt(v_smth));
ucl_x = norminv(0.975, x_smth, sqrt(v_smth));

certainty = 1 - normcdf(prctile(x_smth, 50) * ones(1, length(
    x_smth)), x_smth, sqrt(v_smth));

lcl_p = zeros(1, K);
ucl_p = zeros(1, K);

for k = 1:K
    [lcl_p(k), ucl_p(k)] = get_pk_conf_lims(v_smth(k), b0, x_smth
    (k));
end

fs = 4;
t = (0:(K - 1)) / fs;
tr = ((K - 1):(-1):0) / fs;

u_plot = NaN * ones(1, K);
u_plot(pt) = r(pt);

subplot(511);
hold on;
plot(ty, y, 'k', 'linewidth', 1.25);
ylabel({'(a) skin cond.', '(\mu S)'});
set(gca,'xticklabel', []); ylim([0 3]);
title('State Estimation with Experimental Data'); xlim([0 ty(end)
    ]);
grid;
yl = ylim;

patch([xp(1), xp(2), xp(2), xp(1)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'r', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(2), xp(3), xp(3), xp(2)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'g', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(3), xp(4), xp(4), xp(3)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], [1 0.647059 0], 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(4), xp(5), xp(5), xp(4)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(5), xp(6), xp(6), xp(5)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'y', 'FaceAlpha', 0.2, 'EdgeColor', 'none');

subplot(512);
stem(t, u_plot, 'fill', 'k', 'markersize', 3);
ylabel('(b) n_{k}, r_{k}'); grid; xlim([0 t(end)]); ylim([0 15]);
yl = ylim; set(gca,'xticklabel', []);

patch([xp(1), xp(2), xp(2), xp(1)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'r', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(2), xp(3), xp(3), xp(2)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'g', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(3), xp(4), xp(4), xp(3)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], [1 0.647059 0], 'FaceAlpha', 0.2, 'EdgeColor', 'none');
```

```
patch([xp(4), xp(5), xp(5), xp(4)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(5), xp(6), xp(6), xp(5)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'y', 'FaceAlpha', 0.2, 'EdgeColor', 'none');

subplot(513);
hold on;
plot(t, x_smth, 'b', 'linewidth', 1.25);
fill([t, tr], [lcl_x fliplr(ucl_x)], 'c', 'EdgeColor', 'none', '
    FaceAlpha', 0.5);
ylabel('(c) state (x_{k})');
set(gca,'xticklabel', []); xlim([0 t(end)]);
grid; yl = ylim;

patch([xp(1), xp(2), xp(2), xp(1)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'r', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(2), xp(3), xp(3), xp(2)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'g', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(3), xp(4), xp(4), xp(3)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], [1 0.647059 0], 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(4), xp(5), xp(5), xp(4)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(5), xp(6), xp(6), xp(5)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'y', 'FaceAlpha', 0.2, 'EdgeColor', 'none');

subplot(514);
hold on;
plot(t, p_smth, 'r', 'linewidth', 1.5);
fill([t, tr], [lcl_p fliplr(ucl_p)], [1, 0, (127 / 255)], '
    EdgeColor', 'none', 'FaceAlpha', 0.3);
ylim([0 0.15]);
ylabel('(d) probability (p_{k})');
set(gca,'xticklabel', []); xlim([0 t(end)]);
grid; yl = ylim;

patch([xp(1), xp(2), xp(2), xp(1)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'r', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(2), xp(3), xp(3), xp(2)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'g', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(3), xp(4), xp(4), xp(3)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], [1 0.647059 0], 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(4), xp(5), xp(5), xp(4)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
patch([xp(5), xp(6), xp(6), xp(5)] / fs, [yl(1) yl(1) yl(2) yl(2)
    ], 'y', 'FaceAlpha', 0.2, 'EdgeColor', 'none');

subplot(515);
hold on;
v1 = [0 0.9; t(end) 0.9; t(end) 1; 0 1];
c1 = [1 (220 / 255) (220 / 255); 1 (220 / 255) (220 / 255); 1 0
    0; 1 0 0];
faces1 = [1 2 3 4];
```

```
patch('Faces', faces1, 'Vertices', v1, 'FaceVertexCData', c1, '
    FaceColor', 'interp', ...
    'EdgeColor', 'none', 'FaceAlpha', 0.7);

v2 = [0 0; t(end) 0; t(end) 0.1; 0 0.1];
c2 = [0 0.8 0; 0 0.8 0; (204 / 255) 1 (204 / 255); (204 / 255) 1
    (204 / 255)];
faces2 = [1 2 3 4];

patch('Faces', faces2, 'Vertices', v2, 'FaceVertexCData', c2, '
    FaceColor', 'interp', ...
    'EdgeColor', 'none', 'FaceAlpha', 0.7);

plot(t, certainty, 'color', [(138 / 255) (43 / 255) (226 / 255)],
     'linewidth', 1.5); grid;
ylabel('(d) HAI'); xlabel('time (s)'); xlim([0 t(end)]);



function [y] = get_posterior_mode(x_pred, v_pred, z, r0, r1, vr,
    b0, n)

    M = 100;    % maximum iterations

    it = zeros(1, M);
    f = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)
        if (n == 0)
            C = v_pred;
            f(i) = it(i) - x_pred - C * (n - exp(b0 + it(i)) / (1
     + exp(b0 + it(i))));
            df(i) = 1 + C * exp(b0 + it(i)) / (1 + exp(b0 + it(i)
    )) ^ 2;
        elseif (n == 1)
            C = v_pred / ((r1 ^ 2) * v_pred + vr);
            f(i) = it(i) - x_pred - C * (r1 * (z - r0 - r1 *
    x_pred) + vr * (n - (1 / (1 + exp((-1) * (b0 + it(i)))))));
            df(i) = 1 + C * vr * exp(b0 + it(i)) / ((1 + exp(b0 +
     it(i))) ^ 2);
        end

        it(i + 1)  = it(i) - f(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
            y = it(i + 1);
            return
        end
    end

    error('Newton-Raphson failed to converge.');
```

```
end

function y = get_maximum_variance(z, r0, r1, W, x_smth, pt)

    x_smth = x_smth(pt);
    W = W(pt);
    z = z(pt);
    K = length(pt);

    y = (z * z' + K * (r0 ^ 2) + (r1 ^ 2) * sum(W) ...
                - 2 * r0 * sum(z) - 2 * r1 * dot(x_smth, z) + 2 *
     r0 * r1 * sum(x_smth)) / K;
end

function y = get_linear_parameters(x_smth, W, z, pt)

    x_smth = x_smth(pt);
    W = W(pt);
    z = z(pt);
    K = length(pt);

    y = [K sum(x_smth); sum(x_smth) sum(W)] \ [sum(z); sum(z .*
    x_smth)];

end

function [lcl, ucl] = get_pk_conf_lims(v, b0, x)

    p = (1e-4:1e-4:1);

    fp = cumtrapz(p, 1 ./ (sqrt(2 * pi * v) * p .* (1 - p)) .*
    ...
        exp(((-1) / (2 * v))* (log(p ./ ((1 - p) * exp(b0))) - x)
     .^ 2));

    n = find(fp <= 0.975);
    m = find(fp < 0.025);

    ucl = p(n(end));
    lcl = p(m(end));
end
```

10.6 State-space Model with One MPP and One Continuous Observation

10.6.1 Simulated Data Example

```
load('data_one_mpp_one_cont.mat');

base_prob = sum(n) / length(n);
pt = find(n > 0);

%% parameters

M = 1e6;    % maximum iterations
m = 1;
tol = 1e-8; % convergence criteria

r0 = zeros(1, M);   % continuous model
r1 = zeros(1, M);
vr = zeros(1, M);   % continuous model noise variance (1)

s0 = zeros(1, M);   % continuous model
s1 = zeros(1, M);
vs = zeros(1, M);   % continuous model noise variance (2)

ve = zeros(1, M);   % process noise variance
K = length(n);

x_pred = zeros(1, K);
v_pred = zeros(1, K);

x_updt = zeros(1, K);
v_updt = zeros(1, K);

x_smth = zeros(1, K);
v_smth = zeros(1, K);

p_updt = zeros(1, K);

A = zeros(1, K);
W = zeros(1, K);
CW = zeros(1, K);
C = zeros(1, K);

%% initial guesses

b0 = log(base_prob / (1 - base_prob));
r0(1) = r(1); % guess it's the first value of r
r1(1)  = 0.5;
s0(1) = s(1);
s1(1) = 1;
vr(1) = 0.05;
```

```
vs(1) = 0.05;
ve(1) = 0.05;

%% main function

for m = 1:M

    for k = 1:K

        if (k == 1)
            x_pred(k) = x_smth(1);
            v_pred(k) = ve(m) + ve(m);
        else
            x_pred(k) = x_updt(k - 1);
            v_pred(k) = v_updt(k - 1) + ve(m);
        end

        x_updt(k) = get_posterior_mode(x_pred(k), v_pred(k), r(k)
    , r0(m), r1(m), b0, vr(m), n(k), s(k), s0(m), s1(m), vs(m));
        p_updt(k) = 1 / (1 + exp((-1) * (b0 + x_updt(k))));

        if (n(k) == 0)
            v_updt(k) = 1 / ((1 / v_pred(k)) + ((s1(m) ^ 2) / vs(
    m)) + p_updt(k) * (1 - p_updt(k)));
        elseif (n(k) == 1)
            v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(
    m)) + ((s1(m) ^ 2) / vs(m)) + p_updt(k) * (1 - p_updt(k)));
        end
    end

    x_smth(K) = x_updt(K);
    v_smth(K) = v_updt(K);
    W(K) = v_smth(K) + (x_smth(K) ^ 2);

    A(1:(end - 1)) = v_updt(1:(end - 1)) ./ v_pred(2:end);

    for k = (K - 1):(-1):1
       x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
     1));
       v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
    v_pred(k + 1));

       CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
       W(k) = v_smth(k) + (x_smth(k) ^ 2);
    end

    prev = [r0(m) r1(m) ve(m) vr(m) s0(m) s1(m) vs(m)];

    R = get_linear_parameters_for_mpp(x_smth, W, r, pt);
    S = get_linear_parameters(x_smth, W, s, K);

    ve(m + 1) = (sum(W(2:end)) + sum(W(1:(end - 1))) - 2 * sum(CW
    )) / K;
```

```
    r0(m + 1) = R(1, 1);
    r1(m + 1) = R(2, 1);

    s0(m + 1) = S(1, 1);
    s1(m + 1) = S(2, 1);

    vr(m + 1) = get_maximum_variance_for_mpp(r, r0(m + 1), r1(m +
     1), W, x_smth, pt);
    vs(m + 1) = get_maximum_variance(s, s0(m + 1), s1(m + 1), W,
    x_smth, K);

    next = [r0(m + 1) r1(m + 1) ve(m + 1) vr(m + 1) s0(m + 1) s1(
    m + 1) vs(m +1)];

    mean_dev = mean(abs(next - prev));

    if mean_dev < tol
        fprintf('Converged at m = %d\n\n', m);
        break;
    else
        fprintf('m = %d\nr0 = %.18f\nr1 = %.18f\nvr = %.18f\n\ns0
     = %.18f\ns1 = %.18f\nvs = %.18f\n\nve = %.18f\n\n', ...
        m + 1, r0(m + 1), r1(m + 1), vr(m + 1), s0(m + 1), s1(m +
     1), vs(m + 1), ve(m + 1));

        x_pred = zeros(1, K);
        v_pred = zeros(1, K);

        x_updt = zeros(1, K);
        v_updt = zeros(1, K);

        x_smth(2:end) = zeros(1, K - 1);    % x_smth(1) needed
    for next iteration
        v_smth = zeros(1, K);

        p_updt = zeros(1, K);

        A = zeros(1, K);
        W = zeros(1, K);
        CW = zeros(1, K);
        C = zeros(1, K);
    end
end

%% calculate confidence limits

p_smth = 1 ./ (1 + exp((-1) * (b0 + x_smth)));  % mode, lower and
     upper confidence limits for binary distribution
r_smth = r0(m) + r1(m) * x_smth;
s_smth = s0(m) + s1(m) * x_smth;

r_plot = NaN * ones(1, K);
r_plot(pt) = r(pt);
```

```
%% plot graphs

subplot(511);
hold on;
stem(r_plot, 'fill', 'color', 'b', 'markersize', 4);
plot(r_smth, 'r-.', 'linewidth', 1.25);
ylabel('(a) n_{k}, r_{k}');
title('Estimation with Simulated Data');
grid;

subplot(512);
hold on;
plot(p, 'b');
plot(p_smth, 'r-.', 'linewidth', 1.25);
ylabel('(b) p_{k}');
grid;

subplot(513);
hold on;
plot(s, 'b');
plot(s_smth, 'r-.', 'linewidth', 1.25); grid;
ylabel('(c) s_{k}');

subplot(514);
hold on;
plot(x, 'b');
plot(x_smth, 'r-.', 'linewidth', 1.25); grid;
ylabel('(d) x_{k}'); xlabel('time index');

subplot(515);
qqplot(x - x_smth);
title('QQ Plot - State Estimate', 'FontWeight', 'Normal');
ylabel('(e) input quantiles');
xlabel('standard normal quantiles');
grid;

%% supplementary functions

function y = get_posterior_mode(x_pred, v_pred, r, r0, r1, b0, vr
    , n, s, s0, s1, vs)

    M = 200;    % maximum iterations

    it = zeros(1, M);
    f = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)

        if (n == 0)
            C = v_pred / ((s1 ^ 2) * v_pred + vs);
```

```
            f(i) = it(i) - x_pred - C * (s1 * (s - s0 - s1 *
    x_pred) + vs * (n - (1 / (1 + exp((-1) * (b0 + it(i)))))));
            df(i) = 1 + C * vs * exp(b0 + it(i)) / ((1 + exp(b0 +
     it(i))) ^ 2);
        elseif (n == 1)
            C = v_pred / (vr * vs + v_pred * ((r1 ^ 2) * vs + (s1
     ^ 2) * vr));
            f(i) = it(i) - x_pred - C * (r1 * vs * (r - r0 - r1 *
     x_pred) + s1 * vr * (s - s0 - s1 * x_pred) + ...
                vr * vs * (n - (1 / (1 + exp((-1) * (b0 + it(i)))
    ))));
            df(i) = 1 + C * vr * vs * exp(b0 + it(i)) / ((1 + exp
    (b0 + it(i))) ^ 2);
        end

        it(i + 1) = it(i) - f(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
           y = it(i + 1);
           return;
        end
    end

    error('Newton-Raphson failed to converge.');

end

function y = get_maximum_variance(z, r0, r1, W, x_smth, K)

    y = (z * z' + K * (r0 ^ 2) + (r1 ^ 2) * sum(W) ...
                - 2 * r0 * sum(z) - 2 * r1 * dot(x_smth, z) + 2 *
     r0 * r1 * sum(x_smth)) / K;

end

function y = get_linear_parameters(x_smth, W, z, K)

    y = [K sum(x_smth); sum(x_smth) sum(W)] \ [sum(z); sum(z .*
    x_smth)];

end

function y = get_maximum_variance_for_mpp(z, r0, r1, W, x_smth,
    pt)

    x_smth = x_smth(pt);
    W = W(pt);
    z = z(pt);
    K = length(pt);

    y = (z * z' + K * (r0 ^ 2) + (r1 ^ 2) * sum(W) ...
                - 2 * r0 * sum(z) - 2 * r1 * dot(x_smth, z) + 2 *
     r0 * r1 * sum(x_smth)) / K;
end
```

```
function y = get_linear_parameters_for_mpp(x_smth, W, z, pt)

    x_smth = x_smth(pt);
    W = W(pt);
    z = z(pt);
    K = length(pt);

    y = [K sum(x_smth); sum(x_smth) sum(W)] \ [sum(z); sum(z .*
    x_smth)];

end
```

10.6.2 Experimental Data Example

```
load('expm_data_one_mpp_one_cont.mat');

subj = 1;
T = 1450;
n = zeros(1, T);
r = zeros(1, T);

pt = find(u > 0);
n(pt) = 1;
r(pt) = u(pt);
s = y;

base_prob = sum(n) / length(n);
pt = find(n > 0);

%% parameters

M = 1e6;    % maximum iterations
m = 1;
tol = 1e-8; % convergence criteria

b0 = zeros(1, M);
b1 = zeros(1, M);

r0 = zeros(1, M);   % continuous model
r1 = zeros(1, M);
vr = zeros(1, M);   % continuous model noise variance (1)

s0 = zeros(1, M);   % continuous model
s1 = zeros(1, M);
vs = zeros(1, M);   % continuous model noise variance (2)

ve = zeros(1, M);   % process noise variance
K = length(n);

x_pred = zeros(1, K);
v_pred = zeros(1, K);
```

```
x_updt = zeros(1, K);
v_updt = zeros(1, K);

x_smth = zeros(1, K);
v_smth = zeros(1, K);

p_updt = zeros(1, K);

A = zeros(1, K);
W = zeros(1, K);
CW = zeros(1, K);
C = zeros(1, K);

%% initial guesses

b0(1) = log(base_prob / (1 - base_prob));
b1(1) = 1;
r0(1) = prctile(r(pt), 50);
r1(1)  = 0.5;
s0(1) = s(1);
s1(1) = 1;
vr(1) = 0.05;
vs(1) = 1 * var(s); % 1 * var(s)
ve(1) = 0.05;
lambda = 0.01;  % 0.01

%% main function

for m = 1:M

    for k = 1:K

        if (k == 1)
            x_pred(k) = x_smth(1);
            v_pred(k) = ve(m) + ve(m);
        else
            x_pred(k) = x_updt(k - 1);
            v_pred(k) = v_updt(k - 1) + ve(m);
        end

        x_updt(k) = get_posterior_mode(x_pred(k), v_pred(k), r(k)
    , r0(m), r1(m), b0(m), b1(m), vr(m), n(k), s(k), s0(m), s1(m)
    , vs(m));
        p_updt(k) = 1 / (1 + exp((-1) * (b0(m) + b1(m) * x_updt(k
    ))));

        if (n(k) == 0)
            v_updt(k) = 1 / ((1 / v_pred(k)) + ((s1(m) ^ 2) / vs(
    m)) + (b1(m) ^ 2) * p_updt(k) * (1 - p_updt(k)));
        elseif (n(k) == 1)
            v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(
    m)) + ((s1(m) ^ 2) / vs(m)) + (b1(m) ^ 2) * p_updt(k) * (1 -
    p_updt(k)));
```

```
    end
end

x_smth(K) = x_updt(K);
v_smth(K) = v_updt(K);
W(K) = v_smth(K) + (x_smth(K) ^ 2);

A(1:(end - 1)) = v_updt(1:(end - 1)) ./ v_pred(2:end);

for k = (K - 1):(-1):1
   x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
 1));
   v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
v_pred(k + 1));

   CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
   W(k) = v_smth(k) + (x_smth(k) ^ 2);
end

R = get_linear_parameters_for_mpp(x_smth, W, r, pt);
S = get_linear_parameters(x_smth, W, s, K);

prev = [r0(m) r1(m) ve(m) vr(m) s0(m) s1(m) vs(m) b0(m) b1(m)
];

ve(m + 1) = (sum(W(2:end)) + sum(W(1:(end - 1))) - 2 * sum(CW
)) / K;

bb = fsolve(@(b) binary_parameter_derivatives(b, n, x_smth,
v_smth), [-5 1], optimset('Display','off'));

b0(m + 1) = bb(1);
b1(m + 1) = bb(2);

r0(m + 1) = R(1, 1);
r1(m + 1) = R(2, 1);
vr(m + 1) = get_maximum_variance_for_mpp(r, r0(m + 1), r1(m +
 1), W, x_smth, pt);

if ((vs(m) + lambda * (get_maximum_variance(s, s0(m), s1(m),
W, x_smth, K) - vs(m))) > 0.75 * var(s))  % EM algorithm
intentionally modified slightly for overfitting control
    s0(m + 1) = s0(m) + lambda * (S(1, 1) - s0(m));
    s1(m + 1) = s1(m) + lambda * (S(2, 1) - s1(m));
    vs(m + 1) = vs(m) + lambda * (get_maximum_variance(s, s0(
m), s1(m), W, x_smth, K) - vs(m));
else
    s0(m + 1) = s0(m);
    s1(m + 1) = s1(m);
    vs(m + 1) = vs(m);
end

next = [r0(m + 1) r1(m + 1) ve(m + 1) vr(m + 1) s0(m + 1) s1(
m + 1) vs(m + 1) b0(m + 1) b1(m + 1)];
```

```
    mean_dev = mean(abs(next - prev));

    if (b1(m + 1) < 0) || (r1(m + 1) < 0)   % if this happens
    with experimental data
        fprintf('Iterations halted at m = %d\n\n', m);
        break;
    end

    if mean_dev < tol
        fprintf('Converged at m = %d\n\n', m);
        break;
    else
        fprintf('m = %d\nr0 = %.18f\nr1 = %.18f\nvr = %.18f\n\ns0
     = %.18f\ns1 = %.18f\nvs = %.18f\n\nb0 = %.18f\nb1 = %.18f\n\
    nve = %.18f\n\n', ...
        m + 1, r0(m + 1), r1(m + 1), vr(m + 1), s0(m + 1), s1(m +
     1), vs(m + 1), b0(m + 1), b1(m + 1), ve(m + 1));

        x_pred = zeros(1, K);
        v_pred = zeros(1, K);

        x_updt = zeros(1, K);
        v_updt = zeros(1, K);

        x_smth(2:end) = zeros(1, K - 1);    % x_smth(1) needed
    for next iteration
        v_smth = zeros(1, K);

        p_updt = zeros(1, K);

        A = zeros(1, K);
        W = zeros(1, K);
        CW = zeros(1, K);
        C = zeros(1, K);
    end
end

%% calculate confidence limits

p_smth = 1 ./ (1 + exp((-1) * (b0(m) + b1(m) * x_smth)));  % mode
    , lower and upper confidence limits for binary distribution
r_smth = r0(m) + r1(m) * x_smth;
s_smth = s0(m) + s1(m) * x_smth;

lcl_x = norminv(0.025, x_smth, sqrt(v_smth));
ucl_x = norminv(0.975, x_smth, sqrt(v_smth));

lcl_p = zeros(1, K);
ucl_p = zeros(1, K);

for k = 1:K
    [lcl_p(k), ucl_p(k)] = get_pk_conf_lims(v_smth(k), b0(m), b1(
    m), x_smth(k));
```

```
end

r_plot = NaN * ones(1, K);
r_plot(pt) = r(pt);

%% plot graphs

t = (1:K);
tr = (K:(-1):1);
xtick_pos = 1:(4 * 60):1450;
xtick_labels = {'9 AM', '1 PM', '5 PM', '9 PM', '1 AM', '5 AM', '
    9 AM'};

subplot(411);
hold on;
stem(t, r_plot, 'fill', 'color', 'b', 'markersize', 4);
plot(t, r_smth, 'r-.', 'linewidth', 1.25);
ylabel('(a) n_{k}, r_{k}'); ylim([-inf (max([r_plot, r_smth]) +
    2.5)]);
grid; xlim([0, K]); set(gca, 'xtick', xtick_pos);
set(gca, 'xticklabel', []);
title('State Estimation with Experimental Data');

subplot(412);
hold on;
plot(t, s, 'color', [1 (128 / 255) 0], 'linewidth', 1.25); grid;
plot(t, s_smth, 'r-.', 'linewidth', 1.25);
ylim([0 (max([s, s_smth]) + 2.5)]);
ylabel('(b) s_{k}'); set(gca, 'xtick', xtick_pos);
xlim([0, K]); set(gca, 'xticklabel', []);

subplot(413);
hold on;
col = [0 (176 / 255) (80 / 255)];
fill([t, tr], [lcl_p fliplr(ucl_p)], [(54 / 255) (208 / 255) (80
    / 255)], 'EdgeColor', 'none', 'FaceAlpha', 0.3);
plot(t, p_smth, 'color', [(54 / 255) (150 / 255) (80 / 255)], '
    linewidth', 1.25); grid;
ylabel('(c) p_{k}'); set(gca, 'xtick', xtick_pos); ylim([0 (max(
    ucl_p) + 0.0075)]);
xlim([0, K]); set(gca, 'xticklabel', []);

subplot(414);
hold on;
fill([t, tr], [lcl_x fliplr(ucl_x)], [(102 / 255) 0 (204 / 255)],
     'EdgeColor', 'none', 'FaceAlpha', 0.3);
plot(t, x_smth, 'color', [(102 / 255) 0 (150 / 255)], 'linewidth'
    , 1.25);
grid; xlim([0, K]); ylim([(min(lcl_x) - 1) (max(ucl_x) + 1)]);
set(gca, 'xtick', xtick_pos);
set(gca, 'xticklabel', xtick_labels);
ylabel('(d) x_{k}'); xlabel('time');
```

```
function y = get_posterior_mode(x_pred, v_pred, r, r0, r1, b0, b1
    , vr, n, s, s0, s1, vs)

    M = 200;    % maximum iterations

    it = zeros(1, M);
    f = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)

        if (n == 0)
            C = v_pred / ((s1 ^ 2) * v_pred + vs);
            f(i) = it(i) - x_pred - C * (s1 * (s - s0 - s1 *
    x_pred) + vs * b1 * (n - (1 / (1 + exp((-1) * (b0 + b1 * it(i
    ))))))));
            df(i) = 1 + C * vs * (b1 ^ 2) * exp(b0 + b1 * it(i))
    / ((1 + exp(b0 + b1 * it(i))) ^ 2);
        elseif (n == 1)
            C = v_pred / (vr * vs + v_pred * ((r1 ^ 2) * vs + (s1
     ^ 2) * vr));
            f(i) = it(i) - x_pred - C * (r1 * vs * (r - r0 - r1 *
     x_pred) + s1 * vr * (s - s0 - s1 * x_pred) + ...
                vr * vs * b1 * (n - (1 / (1 + exp((-1) * (b0 + b1
     * it(i)))))));
            df(i) = 1 + C * vr * vs * (b1 ^ 2) * exp(b0 + b1 * it
    (i)) / ((1 + exp(b0 + b1 * it(i))) ^ 2);
        end

        it(i + 1) = it(i) - f(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
           y = it(i + 1);
           return;
        end
    end

    error('Newton-Raphson failed to converge.');

end

function [lcl, ucl] = get_pk_conf_lims(v, b0, b1, x)

    p = (1e-4:1e-4:1);

    fp = cumtrapz(p, 1 ./ (sqrt(2 * pi * v) * b1 * p .* (1 - p))
    .* ...
        exp(((-1) / (2 * v))* ((1 / b1) * log(p ./ ((1 - p) * exp
    (b0))) - x) .^ 2));

    n = find(fp <= 0.975);
    m = find(fp < 0.025);
```

```
    ucl = p(n(end));
    lcl = p(m(end));
end

function y = get_maximum_variance_for_mpp(z, r0, r1, W, x_smth,
    pt)

    x_smth = x_smth(pt);
    W = W(pt);
    z = z(pt);
    K = length(pt);

    y = (z * z' + K * (r0 ^ 2) + (r1 ^ 2) * sum(W) ...
                - 2 * r0 * sum(z) - 2 * r1 * dot(x_smth, z) + 2 *
     r0 * r1 * sum(x_smth)) / K;
end

function y = get_maximum_variance(z, r0, r1, W, x_smth, K)

    y = (z * z' + K * (r0 ^ 2) + (r1 ^ 2) * sum(W) ...
                - 2 * r0 * sum(z) - 2 * r1 * dot(x_smth, z) + 2 *
     r0 * r1 * sum(x_smth)) / K;
end

function y = get_linear_parameters_for_mpp(x_smth, W, z, pt)

    x_smth = x_smth(pt);
    W = W(pt);
    z = z(pt);
    K = length(pt);

    y = [K sum(x_smth); sum(x_smth) sum(W)] \ [sum(z); sum(z .*
    x_smth)];

end

function y = get_linear_parameters(x_smth, W, z, K)

    y = [K sum(x_smth); sum(x_smth) sum(W)] \ [sum(z); sum(z .*
    x_smth)];

end

function y = binary_parameter_derivatives(b, n, x_smth, v_smth)

    y = zeros(1, 2);
    K = length(n);

    b0 = b(1);
    b1 = b(2);
    p = zeros(1, K);

    for k = 1:K
```

```
        p(k) = 1 / (1 + exp((-1) * (b0 + b1 * x_smth(k))));
        y(1) = y(1) + n(k) - p(k) - 0.5 * v_smth(k) * (b1 ^ 2) *
    p(k) * (1 - p(k)) * (1 - 2 * p(k));
        y(2) = y(2) + n(k) * x_smth(k) - x_smth(k) * p(k) - 0.5 *
     v_smth(k) * b1 * p(k) * (1 - p(k)) * (2 + x_smth(k) * b1 *
    (1 - 2 * p(k)));
    end

end
```

10.7 State-space Model with One Binary and One Spiking-type Observation

10.7.1 Experimental Data Example

```
load('expm_data_one_bin_one_spk.mat');

fs = 4;
delta = 0.005;

min_peak_height = 0.1;
min_peak_promn = 0.1;
min_peak_dist = fs;

ph = s.ph;
tn = s.tn;

rpeaks = s.rpeaks;
ul = s.ul;
w = s.w;
theta = s.theta;

[pks, locs] = findpeaks(ph, 'MinPeakHeight', min_peak_height, '
    MinPeakProminence', ...
    min_peak_promn, 'MinPeakDistance', min_peak_dist);

n = zeros(1, length(ph));
n(locs) = 1;

K = length(n);
M = 2e4;
ve = zeros(1, M);   % process noise variance

x_pred = zeros(1, K);
v_pred = zeros(1, K);

x_updt = zeros(1, K);
v_updt = zeros(1, K);

x_smth = zeros(1, K);
```

```
v_smth = zeros(1, K);

p_updt = zeros(1, K);

tpc = 289;  % total (SCR) peak count
tsl = 34182;  % total signal length

base_prob = tpc / tsl;
b0 = log(base_prob / (1 - base_prob));
tol = 5e-8; % convergence criteria

A = zeros(1, K);
W = zeros(1, K);
CW = zeros(1, K);
C = zeros(1, K);

x_smth(1) = 0.44201528159733;
ve(1) = 1.24111644606324e-4;

eta = -0.00004;

exception_counter = 0;

for m = 1:M

    for k = 1:K

        if (k == 1)
            x_pred(k) = x_smth(1);
            v_pred(k) = ve(m) + ve(m);
        else
            x_pred(k) = x_updt(k - 1);
            v_pred(k) = v_updt(k - 1) + ve(m);
        end

        C(k) = v_pred(k);

        try     % numerical issues can occur due to the integrals
            [x_updt(k), H2] = get_posterior_mode(x_pred(k), C(k),
     b0, n(k), rpeaks(k, :), ul(k, :), delta, s.w(k, :, :), theta
    ', eta);
            p_updt(k) = 1 / (1 + exp((-1) * (b0 + x_updt(k))));
            v_updt(k) = 1 / ((1 / v_pred(k)) + p_updt(k) * (1 -
    p_updt(k)) - H2);
        catch
            exception_counter = exception_counter + 1;
            x_updt(k) = x_pred(k);
            v_updt(k) = v_pred(k);
        end

        if (mod(k, 100) == 0)
            fprintf('%d ', k);
        end
```

```
        if (mod(k, 2500) == 0)
            fprintf('\n');
        end

    end

    x_smth(K) = x_updt(K);
    v_smth(K) = v_updt(K);
    W(K) = v_smth(K) + (x_smth(K) ^ 2);

    A(1:(end - 1)) = v_updt(1:(end - 1)) ./ v_pred(2:end);

    for k = (K - 1):(-1):1
       x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
     1));
       v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
    v_pred(k + 1));

       CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
       W(k) = v_smth(k) + (x_smth(k) ^ 2);
    end

    if (m < M)

        ve(m + 1) = (sum(W(2:end)) + sum(W(1:(end - 1))) - 2 *
    sum(CW)) / K;
        mean_dev = mean(abs(ve(m + 1) - ve(m)));

        if mean_dev < tol
            fprintf('m = %d\nx0 = %.18f\nve = %.18f\n\n', m,
    x_smth(1), ve(m));
            fprintf('Converged at m = %d\n\n', m);
            break;
        else
            fprintf('m = %d\nx0 = %.18f\nve = %.18f\n\n', m,
    x_smth(1), ve(m + 1));

            x_pred = zeros(1, K);
            v_pred = zeros(1, K);

            x_updt = zeros(1, K);
            v_updt = zeros(1, K);

            x_smth(2:end) = zeros(1, K - 1);    % x_smth(1)
    needed for next iteration
            v_smth = zeros(1, K);

            p_updt = zeros(1, K);

            A = zeros(1, K);
            W = zeros(1, K);
            CW = zeros(1, K);
            C = zeros(1, K);
        end
```

```
    end
end

p_updt = 1 ./ (1 + exp((-1) * (b0 + x_updt)));
p_smth = 1 ./ (1 + exp((-1) * (b0 + x_smth)));

t = (0:(K - 1)) / (fs * 60);
tr = ((K - 1):(-1):0) / (fs * 60);

lcl_x = norminv(0.025, x_smth, sqrt(v_smth));
ucl_x = norminv(0.975, x_smth, sqrt(v_smth));

lcl_p = zeros(1, K);
ucl_p = zeros(1, K);

for k = 1:K
    [lcl_p(k), ucl_p(k)] = get_pk_conf_lims(v_smth(k), b0, x_smth
    (k));
end

certainty = get_certainty_curve(v_smth, b0, x_smth, base_prob);

lambda = zeros(K, 50);
mean_rr = zeros(K, 50);

for k = 1:K
    for j = 1:50
        w = [squeeze(s.w(k, j, :))' [eta x_smth(k)]];
        if (f(theta', ul(k, j), w) > 1e-18)
           lambda(k, j) = fetch_lambda(theta', ul(k, j), w);
        end
        mean_rr(k, j) =  mu(theta', w);
    end
end

lambda_start_index = find(reshape(rpeaks', 1, numel(rpeaks)), 1);
lambda = reshape(lambda', 1, numel(lambda));
get_ks_plot(find(reshape(rpeaks', 1, numel(rpeaks))) * delta,
    lambda(lambda_start_index:end), delta, 1);

ll = get_log_likelihood(eta, rpeaks, ul, delta, s.w, theta',
    x_smth, v_smth);
ll_final = sum(nansum(ll));
mean_rr = reshape(mean_rr', 1, numel(mean_rr));

rri = diff(s.rpeak_locs);
rr_times = s.rpeak_locs(2:end) / 60;

state_ylim = [(min(lcl_x) - 0.1) (max(ucl_x) + 0.1)];
rr_ylim = [(prctile(rri, 1) - 0.05) (prctile(rri, 99) + 0.05)];
prob_ylim = [(min(lcl_p) - 0.0005) (max(ucl_p(3:end)) + 0.0005)];

figure;
subplot(611);
```

```
hold on;
plot(t, s.x, 'color', [(102 / 255) 0 (204 / 255)]); grid;
set(gca,'xticklabel', []);
ylabel('(a) z_{k}'); xlim([0 t(end)]); ylim([4 22]); title('State
    Estimation with Experimental Data');

subplot(612);
n_plot = NaN * ones(1, K);
n_plot(n > 0) = 1;
stem(t, n_plot, 'fill', 'color', [1, 0, 1], 'markersize', 2);
xlim([0 t(end)]); ylim([0 1.25]);
set(gca,'xticklabel', []);
ylabel('(b) n_{k}'); grid;

subplot(613);
hold on;
plot(t, x_smth, 'b', 'linewidth', 1.25); grid;
set(gca,'xticklabel', []);
fill([t, tr], [lcl_x fliplr(ucl_x)], 'c', 'EdgeColor', 'none', '
    FaceAlpha', 0.2);
ylabel('(c) x_{k}'); xlim([0 t(end)]); ylim(state_ylim);

subplot(614);
hold on;
plot(t, p_smth, 'color', [(102 / 255), 0, (51 / 255)], 'linewidth
    ', 1.25); grid;
set(gca,'xticklabel', []);
fill([t, tr], [lcl_p fliplr(ucl_p)], [1, 0, (127 / 255)], '
    EdgeColor', 'none', 'FaceAlpha', 0.2);
ylabel('(d) p_{k}'); xlim([0 t(end)]); ylim([0.0012 0.0388]);
plot([0, t(end)], [base_prob, base_prob], 'k--', 'linewidth',
    1.25);

subplot(615);
hold on;
plot(rr_times, rri, 'o', 'col', [1, 0.5, 0.25], ...
    'MarkerFaceColor', [1, 0.5, 0.25], 'MarkerSize', 2); grid;
set(gca,'xticklabel', []);
mu_start_index = round(s.rpeak_locs(2) / delta);
plot(((0:(length(mean_rr(mu_start_index:end)) - 1)) * delta) /
    60, mean_rr(mu_start_index:end), 'b');
ylabel('(e) rr_{i}'); xlim([0 t(end)]); ylim(rr_ylim);

subplot(616);
hold on;

v1 = [0 0.9; t(end) 0.9; t(end) 1; 0 1];
c1 = [1 (220 / 255) (220 / 255); 1 (220 / 255) (220 / 255); 1 0
    0; 1 0 0];
faces1 = [1 2 3 4];

patch('Faces', faces1, 'Vertices', v1, 'FaceVertexCData', c1, '
    FaceColor', 'interp', ...
    'EdgeColor', 'none', 'FaceAlpha', 0.7);
```

```
v2 = [0 0; t(end) 0; t(end) 0.1; 0 0.1];
c2 = [0 0.8 0; 0 0.8 0; (204 / 255) 1 (204 / 255); (204 / 255) 1
    (204 / 255)];
faces2 = [1 2 3 4];

patch('Faces', faces2, 'Vertices', v2, 'FaceVertexCData', c2, '
    FaceColor', 'interp', ...
    'EdgeColor', 'none', 'FaceAlpha', 0.7);

plot(t, certainty, 'b', 'linewidth', 1.25); grid; xlim([0 t(end)
    ]);
ylabel('(f) HAI'); xlabel('time (min)'); ylim([0 1]);

function [y, H2] = get_posterior_mode(x_pred, C, b0, n, rpeaks,
    ul, delta, w_all, theta, eta)

    M = 40;    % maximum iterations

    it = zeros(1, M);
    func = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)

        H1 = zeros(1, 50);
        H2 = zeros(1, 50);

        for j = 1:50
            w = [squeeze(w_all(1, j, :))' [eta it(i)]];

            if (f(theta, ul(j), w) > 1e-18) %
                lambda = fetch_lambda(theta, ul(j), w);
                dl_dx = dlambda_dx(theta, ul(j), w);

                H1(j) = dl_dx * (rpeaks(j) - lambda * delta) /
    lambda;
                H2(j) = d2lambda_dx2(theta, ul(j), w) * (rpeaks(j
    ) - lambda * delta) / lambda - rpeaks(j) * (dl_dx ^ 2) / (
    lambda ^ 2);
            end
        end

        H1 = sum(H1);
        H2 = sum(H2);

      func(i) = it(i) - x_pred - C * (n - exp(b0 + it(i)) / (1 +
     exp(b0 + it(i))) + H1);
      df(i) = 1 + C * (exp(b0 + it(i)) / (1 + exp(b0 + it(i))) ^
     2 - H2);
      it(i + 1)  = it(i) - func(i) / df(i);
```

```
        if abs(it(i + 1) - it(i)) < 1e-14
            y = it(i + 1);
           return
        end
    end

    error('Newton-Raphson failed to converge.');

end

function [y] = f(theta, t, w)

    y = sqrt(theta(end) ./ (2 * pi * (t .^ 3))) .* ...
        exp((theta(end) * ((t - mu(theta, w)) .^ 2)) ./ ...
        ((-2) * (mu(theta, w) ^ 2) * t));

end

function [y] = intf(theta, t, w)

    y = integral(@(t)f(theta, t, w), 0, t);

end

function [y] = mu(theta, w)

    eta = w(end - 1);
    x = w(end);
    y = theta(1) + w(1:3) * theta(2:4)' + eta * x;

end

function [y] = fetch_lambda(theta, t, w)

    cdf = intf(theta, t, w);
    y = f(theta, t, w) ./ (1 - cdf);

    if (cdf > 1)    % numerical issue
        y = 0;
    end

end

function [y] = df_dmu(theta, t, w)

    y = (theta(end) / (mu(theta, w) ^ 3)) * (f(theta, t, w) .* (t
     - mu(theta, w)));

end

function [y] = df_dx(theta, t, w)

    eta = w(end - 1);
    y = df_dmu(theta, t, w) .* eta;
```

```
end

function [y] = intdf_dx(theta, t, w)

    y = integral(@(t)df_dx(theta, t, w), 0, t);

end

function [y] = dlambda_dx(theta, t, w)

    cdf = intf(theta, t, w);

    if (cdf > 1)    % numerical issue
        y = 0;
    else
        y = ((1 - cdf) .* df_dx(theta, t, w) + ...
            f(theta, t, w) .* intdf_dx(theta, t, w)) ./ ((1 - cdf
    ) .^ 2);
    end
end

function [y] = d2f_dmu2(theta, t, w)

    y = theta(end) * (df_dmu(theta, t, w) .* ((t - mu(theta, w))
    / (mu(theta, w) ^ 3)) + ...
        f(theta, t, w) .* ((2 * mu(theta, w) - 3 * t) / (mu(theta
    , w) ^ 4)));

end

function [y] = d2f_dx2(theta, t, w)

    eta = w(end - 1);
    y = d2f_dmu2(theta, t, w) .* (eta ^ 2);

end

function [y] = intd2f_dx2(theta, t, w)

    y = integral(@(t)d2f_dx2(theta, t, w), 0, t);

end

function [y] = d2lambda_dx2(theta, t, w)

    y = (2 * dlambda_dx(theta, t, w) * (1 - intf(theta, t, w)) *
    intdf_dx(theta, t, w) + ...
         d2f_dx2(theta, t, w) * (1 - intf(theta, t, w)) + ...
         f(theta, t, w) * intd2f_dx2(theta, t, w)) / ((1 - intf(
    theta, t, w)) ^ 2);

end
```

```
function [y] = get_log_likelihood(eta, rpeaks, ul, delta, w_all,
    theta, x, v)

    K = length(x);
    y = zeros(K, 50);

    for k = 1:K
        for j = 1:50
            w = [squeeze(w_all(k, j, :))' [eta x(k)]];

            if (f(theta, ul(k, j), w) > 1e-18)

                lambda = fetch_lambda(theta, ul(k, j), w);
                dl_dx = dlambda_dx(theta, ul(k, j), w);
                d2l_dx2 = d2lambda_dx2(theta, ul(k, j), w);
                nkj = rpeaks(k, j);

                value = nkj * log(delta * lambda) - delta *
    lambda + ...
                    (d2l_dx2 * (nkj - lambda * delta) / lambda -
    nkj * (dl_dx ^ 2) / (lambda ^ 2)) * v(k) * 0.5;

                if ~isnan(value)
                    y(k, j) = value;
                end
            end

        end
    end
end

function [lcl, ucl] = get_pk_conf_lims(v, b0, x)

    p = (1e-4:1e-4:1);

    fp = cumtrapz(p, 1 ./ (sqrt(2 * pi * v) * p .* (1 - p)) .*
    ...
        exp(((-1) / (2 * v))* (log(p ./ ((1 - p) * exp(b0))) - x)
     .^ 2));

    n = find(fp <= 0.975);
    m = find(fp < 0.025);

    ucl = p(n(end));
    lcl = p(m(end));
end

function certainty = get_certainty_curve(vK, mu, xK, chance_prob)

    p = (1e-4:1e-4:1);
    [~, i] = min(abs(p - chance_prob));
    certainty = zeros(1, length(vK));

    for j = 1:length(vK)
```

```
        fp = cumtrapz(p, 1 ./ (sqrt(2 * pi * vK(j)) * p .* (1 - p
    )) .* ...
             exp(((-1) / (2 * vK(j)))* (log(p ./ ((1 - p) * exp(mu
    ))) - xK(j)) .^ 2));
        certainty(1, j) = 1 - fp(i);
    end
end
```

10.8 State-space Model with One Binary and Two Continuous Observations with a Circadian Input in the State Equation

10.8.1 Experimental Data Example

```
ndays = 5;
T = 1440;
N = ndays * T;
t = (1:N);

load('expm_data_one_bin_two_cont_circadian.mat');

std_r = std(r);
std_s = std(s);

r = r / std_r;
s = s / std_s;

base_prob = sum(n) / length(n);

M = 2e6;
ve = zeros(1, M);   % process noise variance
rho = zeros(1, M);
b0 = zeros(1, M);
b1 = zeros(1, M);

r0 = zeros(1, M);
r1 = zeros(1, M);
vr = zeros(1, M);

s0 = zeros(1, M);
s1 = zeros(1, M);
vs = zeros(1, M);

K = length(n);

x_pred = zeros(1, K);
v_pred = zeros(1, K);

x_updt = zeros(1, K);
v_updt = zeros(1, K);
```

```
x_smth = zeros(1, K);
v_smth = zeros(1, K);

p_updt = zeros(1, K);

tol = 1e-8; % convergence criteria

A = zeros(1, K);
W = zeros(1, K);
CW = zeros(1, K);
C = zeros(1, K);

ve(1) = 0.005;
rho(1) = 0.98;

b0(1) = log(base_prob / (1 - base_prob));
b1(1) = 0.9;

r0(1) = r(1);
r1(1)  = 1;
vr(1) = 0.005;

s0(1) = s(1);
s1(1) = 1;
vs(1) = 0.005;

for m = 1:M

    for k = 1:K

        if (k == 1)
            x_pred(k) = x_smth(1) + I(k);
            v_pred(k) = ve(m) + ve(m);
        else
            x_pred(k) = rho(m) * x_updt(k - 1) + I(k);
            v_pred(k) = (rho(m) ^ 2) * v_updt(k - 1) + ve(m);
        end

        C(k) = v_pred(k) / (vr(m) * vs(m) + v_pred(k) * ((r1(m) ^
     2) * vs(m) + (s1(m) ^ 2) * vr(m)));
        x_updt(k) = get_posterior_mode(x_pred(k), C(k), r(k), r0(
    m), r1(m), b0(m), b1(m), vr(m), n(k), s(k), s0(m), s1(m), vs(
    m));

        p_updt(k) = 1 / (1 + exp((-1) * (b0(m) + b1(m) * x_updt(k
    ))));
        v_updt(k) = 1 / ((1 / v_pred(k)) + ((r1(m) ^ 2) / vr(m))
    + ((s1(m) ^ 2) / vs(m)) + (b1(m) ^ 2) * p_updt(k) * (1 -
    p_updt(k)));

    end

    x_smth(K) = x_updt(K);
```

```
v_smth(K) = v_updt(K);
W(K) = v_smth(K) + (x_smth(K) ^ 2);

A(1:(end - 1)) = rho(m) * v_updt(1:(end - 1)) ./ v_pred(2:end
);

for k = (K - 1):(-1):1
   x_smth(k) = x_updt(k) + A(k) * (x_smth(k + 1) - x_pred(k +
 1));
   v_smth(k) = v_updt(k) + (A(k) ^ 2) * (v_smth(k + 1) -
v_pred(k + 1));

   CW(k) = A(k) * v_smth(k + 1) + x_smth(k) * x_smth(k + 1);
   W(k) = v_smth(k) + (x_smth(k) ^ 2);
end

if (m < M)

    rho(m + 1) = sum(CW) / sum(W(1:end - 1));

    next_ve = (sum(W(2:end)) + (rho(m + 1) ^ 2) * sum(W(1:(
end - 1))) - 2 * rho(m + 1) * sum(CW) - ...
        2 * (I(2:end) * x_smth(2:end)') + 2 * rho(m + 1) * (I
(2:end) * x_smth(1:(end - 1))') + ...
        (I * I')) / K;

    if (next_ve > 0)    % check - in case this happens with
experimental data
        ve(m + 1) =  next_ve;
    else
        ve(m + 1) = ve(m);
    end

    bb = fsolve(@(b) binary_parameter_derivatives(b, n,
x_smth, v_smth), [-5 1], optimset('Display','off'));

    if (bb(2) > 0)    % check - in case this happens with
experimental data
        b0(m + 1) = bb(1);
        b1(m + 1) = bb(2);
    else
        b0(m + 1) = b0(m);
        b1(m + 1) = b1(m);
    end

    a = fminsearch(@(a) circadian_parameters(a, rho(m + 1),
x_smth, t, T), a, optimset('Display', 'off'));
    I = rhythm(a, T, t);

    R = get_linear_parameters(x_smth, W, r, K);
    S = get_linear_parameters(x_smth, W, s, K);

    next_vr = get_continuous_variable_variance_update(r, R(1,
 1), R(2, 1), W, x_smth, K);
```

```
        next_vs = get_continuous_variable_variance_update(s, S(1,
    1), S(2, 1), W, x_smth, K);

        if (abs(next_vr - next_vs) > 0.01)  % overfitting control
    with experimental data
            r0(m + 1) = r0(m);
            r1(m + 1) = r1(m);

            s0(m + 1) = s0(m);
            s1(m + 1) = s1(m);

            vr(m + 1) = vr(m);
            vs(m + 1) = vs(m);
        else
            r0(m + 1) = R(1, 1);
            r1(m + 1) = R(2, 1);

            s0(m + 1) = S(1, 1);
            s1(m + 1) = S(2, 1);

            vr(m + 1) = next_vr;
            vs(m + 1) = next_vs;
        end

        mean_dev = mean(abs([ve(m + 1) rho(m + 1) r0(m + 1) r1(m
    + 1) vr(m + 1) s0(m + 1) s1(m + 1) vs(m + 1) b1(m + 1) b0(m +
    1)] - ...
            [ve(m) rho(m) r0(m) r1(m) vr(m) s0(m) s1(m) vs(m) b1(
    m) b0(m)]));

        if mean_dev < tol
            fprintf('m = %d\nx0 = %.18f\nve = %.18f\nrho = %.18f\
    n\nr0 = %.18f\nr1 = %.18f\nvr = %.18f\ns0 = %.18f\ns1 = %.18f
    \nvs = %.18f\n\nb0 = %.18f\nb1 = %.18f\n\n', ...
                m, x_smth(1), ve(m), rho(m), r0(m), r1(m), vr(m),
    s0(m), s1(m), vs(m), b0(m), b1(m));
            fprintf('Converged at m = %d\n\n', m);
            break;
        else
            fprintf('m = %d\nx0 = %.18f\nve = %.18f\nrho = %.18f\
    n\nr0 = %.18f\nr1 = %.18f\nvr = %.18f\n\ns0 = %.18f\ns1 =
    %.18f\nvs = %.18f\n\nb0 = %.18f\nb1 = %.18f\n\n', m, ...
                x_smth(1), ve(m + 1), rho(m + 1), r0(m + 1), r1(m
    + 1), vr(m + 1), s0(m + 1), s1(m + 1), vs(m + 1), b0(m + 1),
    b1(m + 1));

            x_pred = zeros(1, K);
            v_pred = zeros(1, K);

            x_updt = zeros(1, K);
            v_updt = zeros(1, K);

            x_smth(2:end) = zeros(1, K - 1);    % x_smth(1)
    needed for next iteration
```

```
            v_smth = zeros(1, K);

            p_updt = zeros(1, K);

            A = zeros(1, K);
            W = zeros(1, K);
            CW = zeros(1, K);
            C = zeros(1, K);
        end
    end
end

p_smth = 1 ./ (1 + exp((-1) * (b0(m) + b1(m) * x_smth)));
r_smth = (r0(m) + r1(m) * x_smth) * std_r;
s_smth = (s0(m) + s1(m) * x_smth) * std_s;

index = (0:(K - 1));
t_index = index / (60 * 24);
r_index = ((K - 1):(-1):0); % reverse index
transp = 0.3;

subplot(611);
hold on;
plot(t_index, y, 'color', [(102 / 255) 0 (204 / 255)]); grid;
ylabel('(a) z_{k}');
title('State Estimation with Experimental Data');
xlim([0 t_index(end)]); ylim([0 (1.1 * max(y))]);
set(gca,'xticklabel', []);

subplot(612);
n_plot = NaN * ones(1, K);
n_plot(n > 0) = 1;
stem(t_index, n_plot, 'fill', 'color', [1, 69 / 255, 0], '
    markersize', 2);
xlim([0 t_index(end)]); ylim([0 1.25]);
set(gca,'xticklabel', []);
ylabel('(b) n_{k}'); grid;

subplot(613);
hold on;
plot(t_index, p_smth, 'r', 'linewidth', 1.5); ylim([(0.98 * min(
    p_smth)) (1.08 * max(p_smth))]);
ylabel('(c) p_{k}'); grid;
xlim([0 t_index(end)]);
set(gca,'xticklabel', []);

subplot(614);
hold on;
plot(t_index, r_smth, '--', 'color', [0 0.3 0], 'linewidth', 2);
plot(t_index, r * std_r, 'color', [0 0.9 0]); grid;
xlim([0 t_index(end)]); ylabel('(d) r_{k}');
set(gca,'xticklabel', []);

subplot(615);
```

```
hold on;
plot(t_index, s_smth, '--', 'color', [0.5 (25 / 255) (66 / 255)],
     'linewidth', 2);
plot(t_index, s * std_s, 'color', [1 0.5 (179 / 255)]); grid;
xlim([0 t_index(end)]); ylabel('(e) s_{k}');
set(gca,'xticklabel', []);

subplot(616);
hold on;
plot(t_index, x_smth, 'b', 'linewidth', 1.5);
ylabel('(f) x_{k}');
xlim([0 t_index(end)]); ylim([(min(x_smth) - 1) (max(x_smth) + 1)
    ]);
grid;

xticks(0:0.5:4.5); xticklabels({'0000', '1200', '0000', '1200', '
    0000', '1200', '0000', '1200', '0000', '1200'});
xlabel('time (24h clock)');


function y = get_posterior_mode(x_pred, C, r, r0, r1, b0, b1, vr,
     n, s, s0, s1, vs)

    M = 200;    % maximum iterations

    it = zeros(1, M);
    f = zeros(1, M);
    df = zeros(1, M);

    it(1) = x_pred;

    for i = 1:(M - 1)
        f(i) = it(i) - x_pred - C * (r1 * vs * (r - r0 - r1 *
    x_pred) + s1 * vr * (s - s0 - s1 * x_pred) + ...
            vr * vs * b1 * (n - (1 / (1 + exp((-1) * (b0 + b1 *
    it(i)))))));
        df(i) = 1 + C * vr * vs * (b1 ^ 2) * exp(b0 + b1 * it(i))
     / ((1 + exp(b0 + b1 * it(i))) ^ 2);
        it(i + 1) = it(i) - f(i) / df(i);

        if abs(it(i + 1) - it(i)) < 1e-14
           y = it(i + 1);
           return;
        end
    end

    error('Newton-Raphson failed to converge.');

end

function y = binary_parameter_derivatives(b, n, x_smth, v_smth)

    y = zeros(1, 2);
    K = length(n);
```

```
    b0 = b(1);
    b1 = b(2);
    p = zeros(1, K);

    for k = 1:K
        p(k) = 1 / (1 + exp((-1) * (b0 + b1 * x_smth(k))));
        y(1) = y(1) + n(k) - p(k) - 0.5 * v_smth(k) * (b1 ^ 2) *
    p(k) * (1 - p(k)) * (1 - 2 * p(k));
        y(2) = y(2) + n(k) * x_smth(k) - x_smth(k) * p(k) - 0.5 *
     v_smth(k) * b1 * p(k) * (1 - p(k)) * (2 + x_smth(k) * b1 *
    (1 - 2 * p(k)));
    end

end

function y = get_linear_parameters(x_smth, W, z, K)

    y = [K sum(x_smth); sum(x_smth) sum(W)] \ [sum(z); sum(z .*
    x_smth)];

end

function y = get_continuous_variable_variance_update(z, r0, r1, W
    , x_smth, K)

    y = (z * z' + K * (r0 ^ 2) + (r1 ^ 2) * sum(W) ...
            - 2 * r0 * sum(z) - 2 * r1 * dot(x_smth, z) + 2 * r0
    * r1 * sum(x_smth)) / K;

end

function y = circadian_parameters(a, rho, x_smth, t, T)

    I = rhythm(a, T, t);
    y = (I * I') - 2 * (I(2:end) * x_smth(2:end)') + 2 * rho * (I
    (2:end) * x_smth(1:(end - 1))');
end

function y = rhythm(a, T, t)   % the a0 is ignored
    y = 0 + a(2) * sin(2 * pi * t / T) + a(3) * cos(2 * pi * t /
    T) + ...
        a(4) * sin(2 * pi * t / (T / 2)) + a(5) * cos(2 * pi * t
    / (T / 2));
end
```

Chapter 11
List of Supplementary MATLAB Functions

All the MATLAB code examples accompanying this book can be run directly. The examples are self-contained and do not require additional path variables being set up. The following is a partial list of the supplementary MATLAB functions that are called at various stages by the state estimators.

- get_linear_parameters (. . .)
  Calculates the updates for the constant coefficients (e.g., γ_0 and γ_1) for a continuous variable (e.g., r_k). If this function is present in a MATLAB example where there is an MPP, but not a continuous variable, then it calculates the constant coefficients based on the MPP amplitudes.
- get_maximum_variance (. . .) or get_continuous_variable_variance_update (. . .)
  Calculates the sensor noise variance update (e.g., σ_v^2) for a continuous variable (e.g., r_k). If this function is present in a MATLAB example where there is an MPP, but not a continuous variable, then it calculates the sensor noise variance based on the MPP amplitudes.
- get_linear_parameters_for_mpp (. . .)
  Calculates the updates for the constant coefficients (e.g., γ_0 and γ_1) for a series of MPP amplitudes (e.g., r_k). This function is used to calculate the updates corresponding to an MPP when a continuous variable is also present.
- get_maximum_variance_for_mpp (. . .)
  Calculates the sensor noise variance update (e.g., σ_v^2) for a series of MPP amplitudes. This function is used to calculate the update corresponding to an MPP when a continuous variable is also present.
- get_posterior_mode (. . .) or get_state_update (. . .)
  Calculates the update $x_{k|k}$ based on the Newton–Raphson method
- get_pk_conf_lims (. . .)
  Calculates the confidence limits for the probability of binary event occurrence p_k
- get_certainty_curve (. . .)

D. S. Wickramasuriya, R. T. Faghih, *Bayesian Filter Design for Computational Medicine*, https://doi.org/10.1007/978-3-031-47104-9_11

Calculates the HAI value based on the probability of binary event occurrence p_k exceeding a baseline value

- rhythm (. . .)
  Calculates the cortisol-related circadian term I_k in the state equation
- circadian_parameters (. . .)
  Calculates the log-likelihood term to be optimized when estimating the (cortisol-related) circadian rhythm terms in the state equation
- get_log_likelihood (. . .)
  Calculates the log-likelihood of the term involving the CIF
- get_ks_plot(. . .)
  Calculates the Kolmogorov–Smirnov (KS) plot for assessing the goodness of fit of a CIF to point process observations
- Other functions related to a CIF
  Functions such as fetch_lambda (. . .), dlambda_dx (. . .), f (. . .), and mu (. . .) are all supplementary functions that calculate various components or derivatives related to an HDIG-based CIF

References

1. C. M. McEniery, J. R. Cockcroft, M. J. Roman, S. S. Franklin, and I. B. Wilkinson, "Central blood pressure: current evidence and clinical importance," *European Heart Journal*, vol. 35, no. 26, pp. 1719–1725, 01 2014. [Online]. Available: https://doi.org/10.1093/eurheartj/eht565
2. Z. Ghasemi, C.-S. Kim, E. Ginsberg, A. Gupta, and J.-O. Hahn, "Model-Based Blind System Identification Approach to Estimation of Central Aortic Blood Pressure Waveform From Noninvasive Diametric Circulatory Signals," *Journal of Dynamic Systems, Measurement, and Control*, vol. 139, no. 6, 03 2017, 061003. [Online]. Available: https://doi.org/10.1115/1.4035451
3. R. T. Faghih, "System identification of cortisol secretion: Characterizing pulsatile dynamics," Ph.D. dissertation, Massachusetts Institute of Technology, 2014.
4. A. C. Smith, L. M. Frank, S. Wirth, M. Yanike, D. Hu, Y. Kubota, A. M. Graybiel, W. A. Suzuki, and E. N. Brown, "Dynamic analysis of learning in behavioral experiments," *Journal of Neuroscience*, vol. 24, no. 2, pp. 447–461, 2004.
5. M. J. Prerau, A. C. Smith, U. T. Eden, Y. Kubota, M. Yanike, W. Suzuki, A. M. Graybiel, and E. N. Brown, "Characterizing learning by simultaneous analysis of continuous and binary measures of performance," *Journal of Neurophysiology*, vol. 102, no. 5, pp. 3060–3072, 2009.
6. T. P. Coleman, M. Yanike, W. A. Suzuki, and E. N. Brown, "A mixed-filter algorithm for dynamically tracking learning from multiple behavioral and neurophysiological measures," *The Dynamic Brain: An Exploration of Neuronal Variability and its Functional Significance*, pp. 3–28, 2011.
7. N. Malem-Shinitski, Y. Zhang, D. T. Gray, S. N. Burke, A. C. Smith, C. A. Barnes, and D. Ba, "A separable two-dimensional random field model of binary response data from multi-day behavioral experiments," *Journal of Neuroscience Methods*, vol. 307, pp. 175–187, 2018.
8. A. C. Smith, M. R. Stefani, B. Moghaddam, and E. N. Brown, "Analysis and design of behavioral experiments to characterize population learning," *J. Neurophysiology*, vol. 93, no. 3, pp. 1776–1792, 2005.
9. X. Deng, R. T. Faghih, R. Barbieri, A. C. Paulk, W. F. Asaad, E. N. Brown, D. D. Dougherty, A. S. Widge, E. N. Eskandar, and U. T. Eden, "Estimating a dynamic state to relate neural spiking activity to behavioral signals during cognitive tasks," in *37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2015, pp. 7808–7813.
10. E. N. Brown, L. M. Frank, D. Tang, M. C. Quirk, and M. A. Wilson, "A statistical paradigm for neural spike train decoding applied to position prediction from ensemble firing patterns of rat hippocampal place cells," *Journal of Neuroscience*, vol. 18, no. 18, pp. 7411–7425, 1998.

D. S. Wickramasuriya, R. T. Faghih, *Bayesian Filter Design for Computational Medicine*, https://doi.org/10.1007/978-3-031-47104-9

11. R. Barbieri, L. M. Frank, D. P. Nguyen, M. C. Quirk, V. Solo, M. A. Wilson, and E. N. Brown, "Dynamic analyses of information encoding in neural ensembles," *Neural Computation*, vol. 16, no. 2, pp. 277–307, 2004.
12. M. M. Shanechi, Z. M. Williams, G. W. Wornell, R. C. Hu, M. Powers, and E. N. Brown, "A real-time brain-machine interface combining motor target and trajectory intent using an optimal feedback control design," *PloS One*, vol. 8, no. 4, p. e59049, 2013.
13. M. M. Shanechi, R. C. Hu, M. Powers, G. W. Wornell, E. N. Brown, and Z. M. Williams, "Neural population partitioning and a concurrent brain-machine interface for sequential motor function," *Nature Neuroscience*, vol. 15, no. 12, p. 1715, 2012.
14. M. M. Shanechi, G. W. Wornell, Z. M. Williams, and E. N. Brown, "Feedback-controlled parallel point process filter for estimation of goal-directed movements from neural signals," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 21, no. 1, pp. 129–140, 2012.
15. X. Deng, D. F. Liu, K. Kay, L. M. Frank, and U. T. Eden, "Clusterless decoding of position from multiunit activity using a marked point process filter," *Neural Computation*, vol. 27, no. 7, pp. 1438–1460, 2015.
16. K. Arai, D. F. Liu, L. M. Frank, and U. T. Eden, "Marked point process filter for clusterless and adaptive encoding-decoding of multiunit activity," *bioRxiv*, p. 438440, 2018.
17. A. Yousefi, M. R. Rezaei, K. Arai, L. M. Frank, and U. T. Eden, "Real-time point process filter for multidimensional decoding problems using mixture models," *bioRxiv*, p. 505289, 2018.
18. Y. Yang and M. M. Shanechi, "An adaptive and generalizable closed-loop system for control of medically induced coma and other states of anesthesia," *Journal of Neural Engineering*, vol. 13, no. 6, p. 066019, 2016.
19. Y. Yang, J. T. Lee, J. A. Guidera, K. Y. Vlasov, J. Pei, E. N. Brown, K. Solt, and M. M. Shanechi, "Developing a personalized closed-loop controller of medically-induced coma in a rodent model," *Journal of Neural Engineering*, vol. 16, no. 3, p. 036022, 2019.
20. Y. Yang and M. M. Shanechi, "A generalizable adaptive brain-machine interface design for control of anesthesia," in *37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2015, pp. 1099–1102.
21. M. J. Prerau, K. E. Hartnack, G. Obregon-Henao, A. Sampson, M. Merlino, K. Gannon, M. T. Bianchi, J. M. Ellenbogen, and P. L. Purdon, "Tracking the sleep onset process: an empirical model of behavioral and physiological dynamics," *PLoS Computational Biology*, vol. 10, no. 10, p. e1003866, 2014.
22. R. Barbieri and E. N. Brown, "Application of dynamic point process models to cardiovascular control," *Biosystems*, vol. 93, no. 1–2, pp. 120–125, 2008.
23. ——, "Analysis of heartbeat dynamics by point process adaptive filtering," *IEEE Transactions on Biomedical Engineering*, vol. 53, no. 1, pp. 4–12, 2006.
24. A. Yousefi, I. Basu, A. C. Paulk, N. Peled, E. N. Eskandar, D. D. Dougherty, S. S. Cash, A. S. Widge, and U. T. Eden, "Decoding hidden cognitive states from behavior and physiology using a Bayesian approach," *Neural Computation*, vol. 31, no. 9, pp. 1751–1788, 2019.
25. D. S. Wickramasuriya, C. Qi, and R. T. Faghih, "A state-space approach for detecting stress from electrodermal activity," in *Proc. 40th Annu. Int. Conf. IEEE Eng. Medicine and Biology Society*, 2018.
26. D. S. Wickramasuriya, M. R. Amin, and R. T. Faghih, "Skin conductance as a viable alternative for closing the deep brain stimulation loop in neuropsychiatric disorders," *Frontiers in Neuroscience*, vol. 13, p. 780, 2019.
27. T. Yadav, M. M. Uddin Atique, H. Fekri Azgomi, J. T. Francis, and R. T. Faghih, "Emotional valence tracking and classification via state-space analysis of facial electromyography," in *53rd Asilomar Conference on Signals, Systems, and Computers*, 2019, pp. 2116–2120.
28. M. B. Ahmadi, A. Craik, H. F. Azgomi, J. T. Francis, J. L. Contreras-Vidal, and R. T. Faghih, "Real-time seizure state tracking using two channels: A mixed-filter approach," in *53rd Asilomar Conference on Signals, Systems, and Computers*, 2019, pp. 2033–2039.

29. D. S. Wickramasuriya and R. T. Faghih, "A Bayesian filtering approach for tracking arousal from binary and continuous skin conductance features," *IEEE Transactions on Biomedical Engineering*, vol. 67, no. 6, pp. 1749–1760, 2020.
30. D. S. Wickramasuriya and R. T. Faghih, "A cortisol-based energy decoder for investigation of fatigue in hypercortisolism," in *41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, July 2019, pp. 11–14.
31. D. S. Wickramasuriya and R. T. Faghih, "A mixed filter algorithm for sympathetic arousal tracking from skin conductance and heart rate measurements in Pavlovian fear conditioning," *PloS One*, vol. 15, no. 4, p. e0231659, 2020.
32. D. S. Wickramasuriya and R. T. Faghih, "A marked point process filtering approach for tracking sympathetic arousal from skin conductance," *IEEE Access*, vol. 8, pp. 68 499–68 513, 2020.
33. D. S. Wickramasuriya, S. Khazaei, R. Kiani and R. T. Faghih, "A Bayesian Filtering Approach for Tracking Sympathetic Arousal and Cortisol-related Energy from Marked Point Process and Continuous-valued Observations," *IEEE Access*. https://doi.org/10.1109/ACCESS.2023.3334974.
34. P. J. Soh, G. A. Vandenbosch, M. Mercuri, and D. M.-P. Schreurs, "Wearable wireless health monitoring: Current developments, challenges, and future trends," *IEEE Microwave Magazine*, vol. 16, no. 4, pp. 55–70, 2015.
35. W. Gao, H. Ota, D. Kiriya, K. Takei, and A. Javey, "Flexible electronics toward wearable sensing," *Accounts of Chemical Research*, vol. 52, no. 3, pp. 523–533, 2019.
36. H. F. Azgomi, D. S. Wickramasuriya, and R. T. Faghih, "State-space modeling and fuzzy feedback control of cognitive stress," in *41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2019, pp. 6327–6330.
37. H. F. Azgomi and R. T. Faghih, "A wearable brain machine interface architecture for regulation of energy in hypercortisolism," in *53rd Asilomar Conference on Signals, Systems, and Computers*, 2019, pp. 254–258.
38. R. T. Faghih, M. A. Dahleh, and E. N. Brown, "An optimization formulation for characterization of pulsatile cortisol secretion," *Frontiers in Neuroscience*, vol. 9, p. 228, 2015.
39. H. Taghvafard, M. Cao, Y. Kawano, and R. T. Faghih, "Design of intermittent control for cortisol secretion under time-varying demand and holding cost constraints," *IEEE Transactions on Biomedical Engineering*, vol. 67, no. 2, pp. 556–564, 2019.
40. W. M. Lim, "Demystifying neuromarketing," *Journal of Business Research*, vol. 91, pp. 205–220, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0148296318302716
41. L. Angioletti, F. Cassioli, and M. Balconi, "Neurophysiological correlates of user experience in smart home systems (SHSs): First evidence from electroencephalography and autonomic measures," *Frontiers in Psychology*, vol. 11, p. 411, 2020.
42. E. Whelan, D. McDuff, R. Gleasure, and J. V. Brocke, "How emotion-sensing technology can reshape the workplace," *MIT Sloan Management Review*, vol. 59, no. 3, pp. 7–10, Spring 2018. [Online]. Available: http://search.proquest.com.ezproxy.lib.uh.edu/docview/2023991461?accountid=7107
43. P. A. Low, "Chapter 51 - Sweating," in *Primer on the Autonomic Nervous System (Third Edition)*, 3rd ed., D. Robertson, I. Biaggioni, G. Burnstock, P. A. Low, and J. F. Paton, Eds. San Diego: Academic Press, 2012, pp. 249–251.
44. H. D. Critchley, "Electrodermal responses: what happens in the brain," *The Neuroscientist*, vol. 8, no. 2, pp. 132–142, 2002.
45. M. Benedek and C. Kaernbach, "Decomposition of skin conductance data by means of nonnegative deconvolution," *Psychophysiology*, vol. 47, no. 4, pp. 647–658, 2010.
46. ——, "A continuous measure of phasic electrodermal activity," *Journal of Neuroscience Methods*, vol. 190, no. 1, pp. 80–91, 2010.

47. M. R. Amin and R. T. Faghih, "Sparse deconvolution of electrodermal activity via continuous-time system identification," *IEEE Transactions on Biomedical Engineering*, vol. 66, no. 9, pp. 2585–2595, 2019.
48. R. T. Faghih, P. A. Stokes, M.-F. Marin, R. G. Zsido, S. Zorowitz, B. L. Rosenbaum, H. Song, M. R. Milad, D. D. Dougherty, E. N. Eskandar, A. S. Widge, E. N. Brown, and R. Barbieri, "Characterization of fear conditioning and fear extinction by analysis of electrodermal activity," in *Proc. 37th Annu. Int. Conf. IEEE Eng. Medicine and Biology Society*, 2015, pp. 7814–7818.
49. M. R. Amin and R. T. Faghih, "Inferring autonomic nervous system stimulation from hand and foot skin conductance measurements," in *52nd Asilomar Conference on Signals, Systems, and Computers*, 2018, pp. 655–660.
50. M. R. Amin and R. T. Faghih, "Robust inference of autonomic nervous system activation using skin conductance measurements: A multi-channel sparse system identification approach," *IEEE Access*, vol. 7, pp. 173 419–173 437, 2019.
51. B. Walsh and E. Usler, "Physiological correlates of fluent and stuttered speech production in preschool children who stutter," *Journal of Speech, Language, and Hearing Research*, vol. 62, no. 12, pp. 4309–4323, 2019.
52. P. D. Jong and M. J. Mackinnon, "Covariances for smoothed estimates in state space models," *Biometrika*, vol. 75, no. 3, pp. 601–602, 1988.
53. J. Birjandtalab, D. Cogan, M. B. Pouyan, and M. Nourani, "A non-EEG biosignals dataset for assessment and visualization of neurological status," in *Proc. IEEE Int. Workshop .Signal Processing Systems*, 2016, pp. 110–114.
54. J. A. Healey and R. W. Picard, "Detecting stress during real-world driving tasks using physiological sensors," *IEEE Transactions on intelligent transportation systems*, vol. 6, no. 2, pp. 156–166, 2005.
55. J. A. Russell, "A circumplex model of affect," *Journal of Personality and Social Psychology*, vol. 39, no. 6, p. 1161, 1980.
56. H. J. Pijeira-Díaz, H. Drachsler, S. Järvelä, and P. A. Kirschner, "Sympathetic arousal commonalities and arousal contagion during collaborative learning: How attuned are triad members?" *Computers in Human Behavior*, vol. 92, pp. 188–197, 2019.
57. M.-Z. Poh, N. C. Swenson, and R. W. Picard, "A wearable sensor for unobtrusive, long-term assessment of electrodermal activity," *IEEE Transactions on Biomedical Engineering*, vol. 57, no. 5, pp. 1243–1252, 2010.
58. S. Koelstra, C. Muhl, M. Soleymani, J.-S. Lee, A. Yazdani, T. Ebrahimi, T. Pun, A. Nijholt, and I. Patras, "DEAP: A database for emotion analysis using physiological signals," *IEEE Transactions on Affective Computing*, vol. 3, no. 1, pp. 18–31, 2012.
59. D. R. Bach, G. Flandin, K. J. Friston, and R. J. Dolan, "Modelling event-related skin conductance responses," *Int. J. Psychophysiology*, vol. 75, no. 3, pp. 349–356, 2010.
60. J. J. Braithwaite, D. G. Watson, R. Jones, and M. Rowe, "A guide for analysing electrodermal activity (EDA) & skin conductance responses (SCRs) for psychological experiments," *Psychophysiology*, vol. 49, no. 1, pp. 1017–1034, 2013.
61. M. R. Amin and R. T. Faghih, "Tonic and phasic decomposition of skin conductance data: A generalized-cross-validation-based block coordinate descent approach," in *41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2019, pp. 745–749.
62. A. Greco, G. Valenza, A. Lanata, E. P. Scilingo, and L. Citi, "cvxEDA: A convex optimization approach to electrodermal activity processing," *IEEE Trans. Biomed. Eng.*, vol. 63, no. 4, pp. 797–804, 2016.
63. S. Maren, "Neurobiology of Pavlovian fear conditioning," *Annual Review of Neuroscience*, vol. 24, no. 1, pp. 897–931, 2001.
64. M. R. Milad, S. Igoe, and S. P. Orr, "Fear conditioning in rodents and humans," in *Animal Models of Behavioral Analysis*. Springer, 2011, pp. 111–132.

65. O. V. Lipp, "Human fear learning: Contemporary procedures and measurement," *Fear and Learning: From Basic Processes to Clinical Implications*, no. 2001, pp. 37–51, 2006.
66. D. R. Bach, J. Daunizeau, K. J. Friston, and R. J. Dolan, "PsPM-HRA1: Skin conductance responses in fear conditioning with visual CS and electrical US," Feb. 2017, See the readme file for more detail. Data are stored as .mat files for use with MATLAB in a format readable by the PsPM toolbox (pspm.sourceforge.net). This research was supported by Wellcome Trust grants 098362/Z/12/Z and 098362/Z/12/Z, and Swiss National Science Foundation grant PA00A-117384. [Online]. Available: https://doi.org/10.5281/zenodo.321641
67. ——, "Dynamic causal modelling of anticipatory skin conductance responses," *Biological Psychology*, vol. 85, no. 1, pp. 163–170, 2010.
68. M. Staib, G. Castegnetti, and D. R. Bach, "Optimising a model-based approach to inferring fear learning from skin conductance responses," *J. Neuroscience Methods*, vol. 255, pp. 131–138, 2015.
69. R. C. Drew and L. I. Sinoway, "Autonomic control of the heart," in *Primer on the autonomic nervous system*. Elsevier, 2012, pp. 177–180.
70. J. E. Hall and M. E. Hall, *Guyton and Hall textbook of medical physiology e-Book*. Elsevier Health Sciences, 2020.
71. R. Barbieri, E. C. Matten, A. A. Alabi, and E. N. Brown, "A point-process model of human heartbeat intervals: New definitions of heart rate and heart rate variability," *American Journal of Physiology-Heart and Circulatory Physiology*, vol. 288, no. 1, pp. H424–H435, 2005.
72. R. Barbieri and E. N. Brown, "Analysis of heartbeat dynamics by point process adaptive filtering," *IEEE Trans. Biomed. Eng.*, vol. 53, no. 1, pp. 4–12, 2006.
73. A. L. Mahan and K. J. Ressler, "Fear conditioning, synaptic plasticity and the amygdala: implications for posttraumatic stress disorder," *Trends in Neurosciences*, vol. 35, no. 1, pp. 24–35, 2012.
74. S. Moratti, G. Rubio, P. Campo, A. Keil, and T. Ortiz, "Hypofunction of right temporoparietal cortex during emotional arousal in depression," *Archives of General Psychiatry*, vol. 65, no. 5, pp. 532–541, 2008.
75. M. Soleymani, J. Lichtenauer, T. Pun, and M. Pantic, "A multimodal database for affect recognition and implicit tagging," *IEEE Transactions on Affective Computing*, vol. 3, no. 1, pp. 42–55, 2012.
76. M. K. Abadi, R. Subramanian, S. M. Kia, P. Avesani, I. Patras, and N. Sebe, "DECAF: MEG-based multimodal database for decoding affective physiological responses," *IEEE Transactions on Affective Computing*, vol. 6, no. 3, pp. 209–222, 2015.
77. F. Ringeval, A. Sonderegger, J. Sauer, and D. Lalanne, "Introducing the RECOLA multimodal corpus of remote collaborative and affective interactions," in *Proc. 2013 10th IEEE Int. Conf. and Workshops Automatic Face and Gesture Recognition*, 2013, pp. 1–8.
78. S. Katsigiannis and N. Ramzan, "DREAMER: a database for emotion recognition through EEG and ECG signals from wireless low-cost off-the-shelf devices," *IEEE Journal of Biomedical and Health Informatics*, vol. 22, no. 1, pp. 98–107, 2017.
79. M. De Wied, A. V. Boxtel, J. A. Posthumus, P. P. Goudena, and W. Matthys, "Facial EMG and heart rate responses to emotion-inducing film clips in boys with disruptive behavior disorders," *Psychophysiology*, vol. 46, no. 5, pp. 996–1004, 2009.
80. L. Kulke, D. Feyerabend, and A. Schacht, "A comparison of the Affectiva iMotions facial expression analysis software with EMG for identifying facial expressions of emotion," *Frontiers in Psychology*, vol. 11, p. 329, 2020.
81. B. t Hart, M. E. Struiksma, A. van Boxtel, and J. J. Van Berkum, "Emotion in stories: Facial EMG evidence for both mental simulation and moral evaluation," *Frontiers in psychology*, vol. 9, p. 613, 2018.
82. D. S. Wickramasuriya and R. T. Faghih, "Online and offline anger detection via electromyography analysis," in *IEEE Healthcare Innovations and Point of Care Technologies (HI-POCT)*, 2017, pp. 52–55.

83. R. D. Lane, K. McRae, E. M. Reiman, K. Chen, G. L. Ahern, and J. F. Thayer, "Neural correlates of heart rate variability during emotion," *Neuroimage*, vol. 44, no. 1, pp. 213–222, 2009.
84. H. Nakahara, S. Furuya, S. Obata, T. Masuko, and H. Kinoshita, "Emotion-related changes in heart rate and its variability during performance and perception of music," *Annals of the New York Academy of Sciences*, vol. 1169, no. 1, pp. 359–362, 2009.
85. B. M. Appelhans and L. J. Luecken, "Heart rate variability as an index of regulated emotional responding," *Review of General Psychology*, vol. 10, no. 3, pp. 229–240, 2006.
86. J. Zhu, L. Ji, and C. Liu, "Heart rate variability monitoring for emotion and disorders of emotion," *Physiological measurement*, vol. 40, no. 6, p. 064004, 2019.
87. A. S. Ravindran, S. Nakagome, D. S. Wickramasuriya, J. L. Contreras-Vidal, and R. T. Faghih, "Emotion recognition by point process characterization of heartbeat dynamics," in *IEEE Healthcare Innovations and Point of Care Technologies,(HI-POCT)*. IEEE, 2019, pp. 13–16.
88. H. Takase and Y. Haruki, "Coordination of breathing between ribcage and abdomen in emotional arousal," in *Respiration and Emotion*. Springer, 2001, pp. 75–86.
89. A. Umezawa, "Facilitation and inhibition of breathing during changes in emotion," in *Respiration and emotion*. Springer, 2001, pp. 139–148.
90. C.-K. Wu, P.-C. Chung, and C.-J. Wang, "Representative segment-based emotion analysis and classification with automatic respiration signal segmentation," *IEEE Transactions on Affective Computing*, vol. 3, no. 4, pp. 482–495, 2012.
91. P. Gomez and B. Danuser, "Relationships between musical structure and psychophysiological measures of emotion." *Emotion*, vol. 7, no. 2, p. 377, 2007.
92. D. S. Wickramasuriya, M. K. Tessmer, and R. T. Faghih, "Facial expression-based emotion classification using electrocardiogram and respiration signals," in *IEEE Healthcare Innovations and Point of Care Technologies,(HI-POCT)*, 2019, pp. 9–12.
93. M. Balconi, E. Grippa, and M. E. Vanutelli, "What hemodynamic (fNIRS), electrophysiological (EEG) and autonomic integrated measures can tell us about emotional processing," *Brain and Cognition*, vol. 95, pp. 67–76, 2015.
94. E. Glotzbach, A. Mühlberger, K. Gschwendtner, A. J. Fallgatter, P. Pauli, and M. J. Herrmann, "Prefrontal brain activation during emotional processing: a functional near infrared spectroscopy study (fNIRS)," *The Open Neuroimaging Journal*, vol. 5, p. 33, 2011.
95. M. Balconi, E. Grippa, and M. E. Vanutelli, "Resting lateralized activity predicts the cortical response and appraisal of emotions: An fNIRS study," *Social Cognitive and Affective Neuroscience*, vol. 10, no. 12, pp. 1607–1614, 2015.
96. X. Hu, C. Zhuang, F. Wang, Y.-J. Liu, C.-H. Im, and D. Zhang, "fNIRS evidence for recognizably different positive emotions," *Frontiers in Human Neuroscience*, vol. 13, p. 120, 2019.
97. S. Parshi, R. Amin, H. F. Azgomi, and R. T. Faghih, "Mental workload classification via hierarchical latent dictionary learning: A functional near infrared spectroscopy study," in *IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, 2019, pp. 1–4.
98. M. S. Seet, M. R. Amin, N. I. Abbasi, J. Hamano, A. Chaudhury, A. Bezerianos, R. T. Faghih, and A. Dragomir, "Olfactory-induced positive affect and autonomic response as a function of hedonic and intensity attributes of fragrances," in *42nd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2020.
99. U. T. Eden, L. Srinivasan, and S. V. Sarma, "Neural signal processing tutorial II: Point process model estimation and goodness-of-fit analysis," in *Neural Signal Processing: Quantitative Analysis of Neural Activity*, P. Mitra, Ed. Washington DC: Society for Neuroscience, 2008, ch. 9, pp. 79–87.
100. A. Tzovara, N. Hofer, D. R. Bach, G. Castegnetti, S. Gerster, C. W. Korn, P. C. Paulus, and M. Staib, "PsPM-TC: SCR, ECG, EMG and respiration measurements in a discriminant trace fear conditioning task with visual CS and electrical US." Aug. 2018, See the readme file for more detail. Data are stored as .mat files for use with MATLAB (The MathWorks Inc.,

Natick, USA) in a format readable by the PsPM toolbox (pspm.sourceforge.net). All Matlab files are saved in MATLAB 9.2 (R2017a) format. This research was supported by Wellcome Trust grant 091593/Z/10/Z, and Swiss National Science Foundation grant 320030_149586/1. [Online]. Available: https://doi.org/10.5281/zenodo.1404810

101. S. Melmed, K. S. Polonsky, P. R. Larsen, and H. M. Kronenberg, *Williams Textbook of Endocrinology E-Book*. Elsevier Health Sciences, 2015.
102. A. Clow, F. Hucklebridge, and L. Thorn, "The cortisol awakening response in context," in *Science of Awakening*, ser. International Review of Neurobiology, A. Clow and L. Thorn, Eds. Academic Press, 2010, vol. 93, pp. 153–175. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0074774210930079
103. E. K. Do Yup Lee and M. H. Choi, "Technical and clinical aspects of cortisol as a biochemical marker of chronic stress," *BMB Reports*, vol. 48, no. 4, p. 209, 2015.
104. P. M. Stewart and J. D. Newell-Price, "Chapter 15 - The adrenal cortex," in *Williams Textbook of Endocrinology (Thirteenth Edition)*, thirteenth ed., S. Melmed, K. S. Polonsky, P. R. Larsen, and H. M. Kronenberg, Eds., Philadelphia, 2016, pp. 489–555. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780323297387000150
105. R. T. Faghih, K. Savla, M. A. Dahleh, and E. N. Brown, "A feedback control model for cortisol secretion," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2011, pp. 716–719.
106. R. T. Faghih, M. A. Dahleh, G. K. Adler, E. B. Klerman, and E. N. Brown, "Quantifying pituitary-adrenal dynamics and deconvolution of concurrent cortisol and adrenocorticotropic hormone data by compressed sensing," *IEEE Transactions on Biomedical Engineering*, vol. 62, no. 10, pp. 2379–2388, 2015.
107. ——, "Deconvolution of serum cortisol levels by using compressed sensing," *PloS one*, vol. 9, no. 1, p. e85204, 2014.
108. C. Kirschbaum and D. H. Hellhammer, "Salivary cortisol," *Encyclopedia of stress*, vol. 3, no. 379–383, 2000.
109. M. Anitescu, H. T. Benzon, and R. Variakojis, "Chapter 44 - Pharmacology for the interventional pain physician," in *Practical Management of Pain (Fifth Edition)*, 5th ed., H. T. Benzon, J. P. Rathmell, C. L. Wu, D. C. Turk, C. E. Argoff, and R. W. Hurley, Eds. Philadelphia: Mosby, 2014, pp. 596–614.e4. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B978032308340900044X
110. T. Kuo, A. McQueen, T.-C. Chen, and J.-C. Wang, "Regulation of glucose homeostasis by glucocorticoids," in *Glucocorticoid Signaling*. Springer, 2015, pp. 99–126.
111. L. Rui, "Energy metabolism in the liver," *Comprehensive Physiology*, vol. 4, no. 1, pp. 177–197, 2011.
112. D. S. Wickramasuriya, L. J. Crofford, A. S. Widge, and R. T. Faghih, "Hybrid decoders for marked point process observations and external influences," *IEEE Transactions on Biomedical Engineering*, vol. 70, no. 1, pp. 343–353, 2023.
113. D. D. Pednekar, M. R. Amin, H. F. Azgomi, K. Aschbacher, L. J. Crofford, and R. T. Faghih, "Characterization of cortisol dysregulation in fibromyalgia and chronic fatigue syndromes: A state-space approach," *IEEE Transactions on Biomedical Engineering*, 2020.
114. ——, "A system theoretic investigation of cortisol dysregulation in fibromyalgia patients with chronic fatigue," in *41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2019, pp. 6896–6901.
115. M. M. Shanechi, J. J. Chemali, M. Liberman, K. Solt, and E. N. Brown, "A brain-machine interface for control of medically-induced coma," *PLoS Computational Biology*, vol. 9, no. 10, p. e1003284, 2013.
116. R. G. Krishnan, U. Shalit, and D. Sontag, "Structured inference networks for nonlinear state space models," in *31st AAAI Conf. Artificial Intelligence*, 2017.
117. X. Zheng, M. Zaheer, A. Ahmed, Y. Wang, E. P. Xing, and A. J. Smola, "State space LSTM models with particle MCMC inference," *arXiv preprint arXiv:1711.11179*, 2017.
118. D. S. Wickramasuriya and R. T. Faghih, "A novel filter for tracking real-world cognitive stress using multi-time-scale point process observations," in *41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, July 2019, pp. 599–602.

119. S. Koldijk, M. Sappelli, S. Verberne, M. A. Neerincx, and W. Kraaij, "The SWELL knowledge work dataset for stress and user modeling research," in *16th International Conference on Multimodal Interaction*. ACM, 2014, pp. 291–298.
120. E. N. Brown, P. M. Meehan, and A. P. Dempster, "A stochastic differential equation model of diurnal cortisol patterns," *American Journal of Physiology-Endocrinology and Metabolism*, vol. 280, no. 3, pp. E450–E461, 2001.
121. I. Vargas, A. N. Vgontzas, J. L. Abelson, R. T. Faghih, K. H. Morales, and M. L. Perlis, "Altered ultradian cortisol rhythmicity as a potential neurobiologic substrate for chronic insomnia," *Sleep Medicine Reviews*, vol. 41, pp. 234–243, 2018.
122. D. M. Arble, G. Copinschi, M. H. Vitaterna, E. Van Cauter, and F. W. Turek, "Chapter 12 - Circadian rhythms in neuroendocrine systems," in *Handbook of Neuroendocrinology*, G. Fink, D. W. Pfaff, and J. E. Levine, Eds. San Diego: Academic Press, 2012, pp. 271–305. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780123750976100125
123. F. Suay and A. Salvador, "Chapter 3 - Cortisol," in *Psychoneuroendocrinology of Sport and Exercise: Foundations, Markers, Trends*, F. Ehrlenspiel and K. Strahler, Eds. Routledge, 2012, pp. 43–60.
124. M. A. Lee, N. Bakh, G. Bisker, E. N. Brown, and M. S. Strano, "A pharmacokinetic model of a tissue implantable cortisol sensor," *Advanced Healthcare Materials*, vol. 5, no. 23, pp. 3004–3015, 2016.
125. H. Raff and T. Carroll, "Cushing's syndrome: From physiological principles to diagnosis and clinical care," *The Journal of Physiology*, vol. 593, no. 3, pp. 493–506, 2015.
126. M. N. Starkman and D. E. Schteingart, "Neuropsychiatric manifestations of patients with Cushing's syndrome: relationship to cortisol and adrenocorticotropic hormone levels," *Archives of Internal Medicine*, vol. 141, no. 2, pp. 215–219, 1981.
127. R. A. Feelders, S. Pulgar, A. Kempel, and A. Pereira, "The burden of Cushing's disease: Clinical and health-related quality of life aspects," *European Journal of Endocrinology*, vol. 167, no. 3, pp. 311–326, 2012.
128. A. Lacroix, R. A. Feelders, C. A. Stratakis, and L. K. Nieman, "Cushing's syndrome," *The Lancet*, vol. 386, no. 9996, pp. 913–927, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140673614613751

Index

D. S. Wickramasuriya, R. T. Faghih, *Bayesian Filter Design for Computational Medicine*, https://doi.org/10.1007/978-3-031-47104-9