We, the authors, vividly remember our first 5 years of coding with POM, specifically writing code to analyze data. Scientific programming is rather different from coding for say, building a graphical user interface (GUI). Everything gets more complicated when data get involved and we were admittedly rather lost during that entire time, writing convoluted code where programs included thousands of lines, code that was impossible to maintain or understand even after a short period of time. In essence, we were lacking principles of software development design for the analysis of data.

Since that time, we have hit on principles that work and we will outline and detail them in this chapter. It has not escaped our notice that these principles closely resemble what seems to have been implemented by the perceptual system (at least in the primate, to the degree of understanding that we have now). This makes sense: perceptual systems—we will show this with the example of the visual system because it is (to date) the most studied and perhaps best understood—are designed to analyze the environment for the extraction of relevant actionable information. They have been in development for hundreds of millions of years under relentless evolutionary pressure, yielding a high-performance analysis framework.

As far as we can tell, all sensory systems (with the exception of olfaction, which is special) follow the following five steps, and there are principled reasons for this (Fig. 4.1).

Step 1: Transduction. Every sensory system has to convert some kind of physical energy in the environment into the common currency of the brain. This currency is action potentials or spikes. In the case of vision, photons enter the eye through a narrow aperture (the pupil) and are focused on the retina by the lens. The retina transduces (i.e., converts) the physical energy of photons into action potentials. What leaves the eye is a train of action potentials (a spike train, see chapter: Wrangling Spikes Trains), carried by the optic nerve. The code equivalent of this is to write dedicated "loader" code. Its purpose is to load data from whatever format it was logged in—each physiological data recording

DOI: http://dx.doi.org/10.1016/B978-0-12-804043-0.00004-0

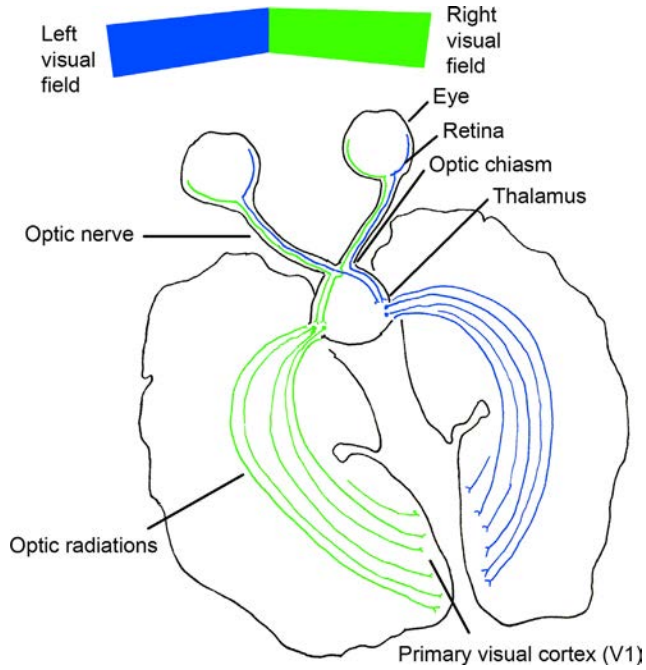

FIGURE 4.1 A cartoon of the primate visual system from the eye to V1. We omitted quite a few structures here, e.g., the superior colliculus or the suprachiasmatic nucleus, but this is the basic signal flow.

system creates its own data files, e.g., .plx files, .nev files, .nex files, .xls files, and .txt files are some popular formats. For you to be able to do anything with these in POM, they have to be converted into a format that POM can use first.

Step 2: Filtering. Once a spike train reaches the cortex, it will be processed. This can be a problem if it corresponds to information that is either not relevant at the time or if it is malformed in some way. In other words, the brain needs a gatekeeper to keep this irrelevant information out. In the brain, this step corresponds to the thalamus, specifically the lateral-geniculate nucleus in the visual system. This nucleus relays information from the retina to the visual cortex, but does so in a selective fashion. Of course this raises the issue of how the thalamus knows what irrelevant information is before it has been analyzed by the cortex. The brain solves this in several ways, including an analysis for low-level

salient features, such as fast motion or high contrast, and then feeds that information back to the thalamus—there are many recurrent feedback loops to fine-tune the filtering process. We strongly recommend to implement this step in code, for similar reasons. It is very hard to write analysis code that is flexible enough to handle data that are not usable, be it due to parts of the data being missing, the study participant not doing what they are supposed to, the data being corrupted, the electrode not working, or the like. If such data enter the processing cascade, it usually renders the result meaningless or breaks the code altogether. It is best to avoid such data being processed in the first place. This step can be called "cleaning," "pruning," or "filtering" of the data. It is important that this step is performed agnostic to the results of the analysis. If you throw out data that don't conform to your hypothesis, you can get data supporting any hypothesis (this is data doctoring, which is unacceptable), don't do it. In contrast, this kind of integrity check before doing the full-blown analysis is critical. Another analogy for this step is that of a gatekeeper—the CEO (the cortex) can't be bothered with irrelevant information. There is too much information out there—if all of that was let in, nothing would get done. That's where a strict personal assistant comes in. Make sure to implement that in code.

Step 3: Formatting. The next step performed by the visual system is a categorical analysis of the data arriving from the thalamus. This step is performed by the early visual system, particularly V1. Here, the visual system determines the location and basic orientation of line segments (Hubel & Wiesel, 2004), and starts the process of determining what is foreground and what is background figure (Craft, Schütze, Niebur, & Von Der Heydt, 2007). Heuristically, this step can be understood as setting the stage for further analysis, not so much doing a lot of in depth analysis here already. The reasons for this will be understood more clearly when discussing the next step. Thus, we conceive of this step as "formatting" the data for further analysis. It is an absolutely critical step for data analysis. Once data are formatted properly, the rest of the analysis is usually rather straightforward. It might be unsettling to the beginner, but is not unusual to spend *most* of one's time writing analysis code in this step, simply "formatting" the data. Once data structures are set up properly, the actual analysis often corresponds to something very simple, like "loop through all participants in these conditions, then compare their means." Similarly, the visual system recognizes the importance of this step—the "early" visual system (V1 and V2) makes up about half the visual system by area in the primate (Wallisch, 2014).

Step 4: Computations. In the visual system, this step is implemented by extrastriate cortex—the cortical regions after striate cortex (or primary visual cortex) in the visual processing stream. Interestingly, whereas the previous steps have been done mostly in serial fashion, one after the other (the feedback to thalamus notwithstanding), this step is better referred to as steps (plural) because they happen in *parallel,* meaning that the signal might split into two or more copies so that multiple processes can occur on it simultaneously (Wallisch & Movshon, 2008). The fundamental reason for this is that many computations, in order to achieve the goal of the computation, have to abstract from some aspects of the source information, in effect destroying it. This might be information that is also important, but is better computed (in parallel) by another area. In effect, different parts of extrastriate cortex (at a minimum dorsal

and ventral stream; Ungerleider & Mishkin, 1983) make copies of the information provided by V1 and work on that toward some outcome. For instance, in order to compute the speed of objects, it might be necessary to abstract from their location and identity, information that is also crucial to the organism, but can't be computed at the same time by the same area or serially. A parallel approach—working off copies of the original information—is perhaps the best way to solve this problem. We recommend to do something similar in code to implement this step. Specifically, we recommend to create as many parallel analysis streams as there are analysis goals. The number of analysis goals is given by how many theoretical questions need to be answered for any given project. For instance, it is conceivable that one analysis is concerned with the mean response of neurons under certain conditions whereas another deals with its variability—the underlying analyses are best done on copies of the original dataset and are complementary to each other. More analyses might build on these, e.g., a correlational analysis, as we will attempt here. We recommend to label these steps 4a, 4b, 4c, etc., in the code, signifying analysis steps that are in principle independent of each other, but can rely on each other, e.g., 4c being executed after 4a, etc. Note that *parallel processing* has a similar meaning in computer science, where computations are performed on data split onto different machines (Fig. 4.2).

Step 5: Output. This might come as a surprise to people living in the modern age, but the purpose of the visual system is not to provide fancy images for one's viewing pleasure, but to improve the survivability of the organism. This is true for sensory systems in general. Perception is not an end in itself—unless it results in motor output, the outcomes of its calculations are irrelevant. Over time, the system has been optimized to provide more adaptive outputs. In primates, the end result of the visual processing cascade in extrastriate areas is linked to motor cortex in order to transform the visual input to real-world outputs and memory systems to store information (the results of the computations on the inputs). We will do the same here, in code. Specifically, we will hook up the outputs of step 4, e.g., 4a, 4b to corresponding outputs, i.e., 5a, 5b. This analogy is tight. There are three principal kinds of outputs from the computation steps: Sometimes, we just want to output some numbers to the screen. Sometimes, the output will be a figure that visualizes the results of a computation. Usually, we also want to store the results so that we can load them in later without having to redo the entire analysis from scratch (this is particularly important as some analyses can take rather long). So just like the brain, we interface with long term memory systems at this point. In the case of code, this is the hard disk of your machine (or—rather soon—likely the cloud). You will want to have as many matching output functions (or files) as there are outputs of step 4.

And that's it. This is the general purpose framework ("the canonical data analysis cascade") that can be used for any data analysis project. We believe it is efficient and use it ourselves on a daily basis. As far as we can tell, most sensory systems do as well. Note that sometimes, some of these steps can be combined into one, e.g., we'll attempt to do the pruning/cleaning step at the same time as the formatting step in order to realize further efficiency gains. Sometimes you'll want to combine calculation and output steps, although one usually can output the same information in multiple

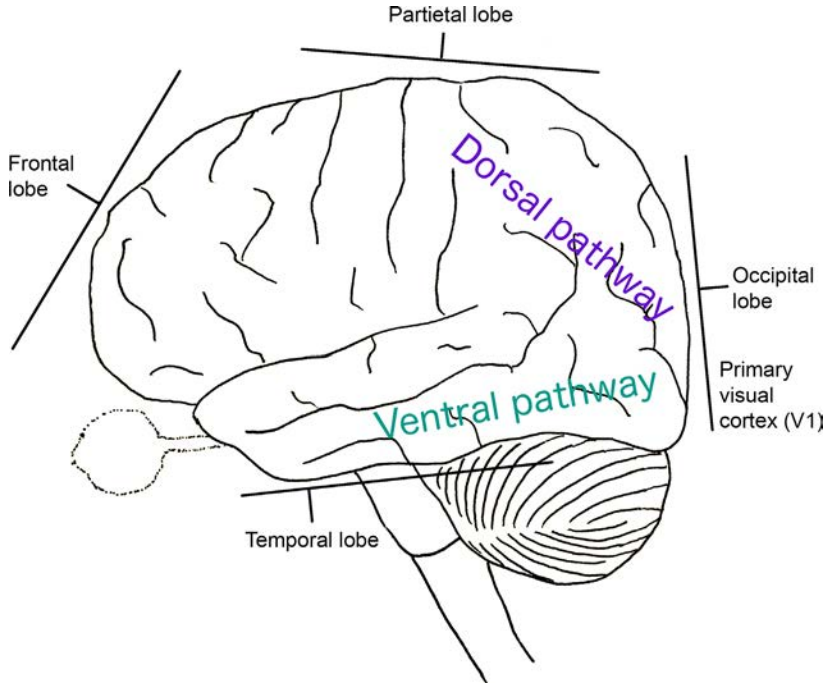

FIGURE 4.2 A cartoon of the extrastriate visual system beyond V1. It's endpoints interface with associative areas in the parietal lobe which are connected to motor systems in the frontal lobe and with memory systems in the temporal lobe.

ways. As long as you are careful when doing so, there is no problem with that, although we recommend separating the steps more strictly if you are an absolute novice until all of this becomes second nature.

To conclude, we strongly advise to recreate these five principal steps in code, when attempting any full-scale data analysis project. Specifically, we recommend to partition analysis code into these steps, i.e., either by writing individual files that correspond to each step or by dividing the code up into self-contained segments. What is important is that each logical segment fits on a screen (or just about fits), if it does not, it will be very hard to maintain. In our experience, analysis code has to be revisited with surprising regularity (e.g., when reviewer 2 asks for additional analyses) and unfortunately, memory for code seems to be particularly ephemeral transient. In other words, if you do not organize and comment your

code well, you will not understand the code you wrote, even after a surprisingly short time interval. This can put you in a difficult position, for instance when pressed for time, as in looming grant or conference deadlines. Avoid it.

In addition to these five steps implemented by sensory systems of the brain, we recommend adding a zeroth and a sixth step. The zeroth step is an "initialization" step (in the brain, this might correspond to birth) you want to start from as nearly blank a slate as possible. In our experience, many logical errors in programming are due to something still lingering in memory that you forgot about but that affects the execution of your code. These errors are hard to track down and can be catastrophic. It is best to avoid them and the best way to do that is to clear the memory before doing anything else. In addition, you will want to define some constants, parameters, or analysis flags in this step. The reason why you want to do this here (and not strewn throughout the code) is that you have an easily accessible section—all in one place, at the beginning of the code—that governs the execution of the code, e.g., options to run the analysis with or without normalization, setting which baseline to use and things like that. Even the brain (at birth) doesn't start with a completely blank slate (due to the complex nature of the development of synaptic organization via both genetics and maternal environment). There is a reason for that.

Finally, in the spirit of making the code maintainable, and even runable a couple of months after writing it, we recommend writing a file that corresponds to the sixth step, which is kind of a "wrapper" and "readme" file that (if you wrote a file for each of the five steps) calls the right files with the right parameters in the right order and some kind of documentation of what is going on. This superstructure perhaps corresponds to the brain as a whole. Although we haven't managed to find the documentation yet.