Signal Detection Theory with Ambivalent or Missing Responses

Samuel R. Mathias, Emma E. E. M. Knowles, and David C. Glahn

Yale University

Author Note

Samuel R. Mathias, Department of Psychiatry, Yale School of Medicine, Yale University, New Haven, Connecticut; Emma E. E. M. Knowles, Department of Psychiatry, Yale School of Medicine, Yale University, New Haven, Connecticut; David C. Glahn, Department of Psychiatry, Yale School of Medicine, Yale University, New Haven, Connecticut.

This work was supported in part by grants from the National Institute of Mental Health.

Correspondence concerning this article should be addressed to Samuel R. Mathias, Suite 3014, 2 Church Street South New Haven, CT 06519.

E-mail: samuel.mathias@yale.edu

Abstract

Signal detection theory (SDT) allows researchers to calculate psychologically meaningful measures from the counts of responses and trials in behavioral experiments. Most SDT models require that subjects make an affirmative response (e.g., “yes” or “no;” “first” or “second”) to each trial in the experiment. Occasionally, researchers may wish to allow subjects to make ambivalent responses (e.g., “I don’t know”), or not force them to respond to all trials. Here, we propose modifications to the standard SDT models that allow ambivalence and omissions in yes-no and two-alternative forced-choice experiments. Under the modified models, equations for calculating sensitivity are the same as those of regular SDT, but equations for calculating bias are different. Furthermore, the modified models include additional psychologically meaningful measures that allow researchers to quantify ambivalence or uncertainty in the decision process.

Keywords: signal detection theory, methods, modeling, statistics

Signal Detection Theory with Ambivalent or Missing Responses

Introduction

Signal detection theory (SDT) is a widely used framework that allows researchers to transform the raw data from an experiment—the counts of trials and responses—into psychologically meaningful measures of sensitivity and bias. Methods for calculating these measures are generally straightforward to implement for common experimental designs (see Green & Swets, 1966; Macmillan & Creelman, 2005). The two most common designs are yes-no (YN) and two-alternative forced-choice (2AFC).

In most experiments with SDT-friendly designs, subjects are required make an affirmative response (e.g., “yes” or “no;” “first” or “second”) to each trial. However, researchers occasionally may wish to conduct experiments that do not require affirmative responses. For example, the standard YN and 2AFC designs could be modified to include an additional response that allows subjects to indicate ambivalence towards the other responses (e.g. “I don’t know”). However, including ambivalent responses flouts the requirements of regular SDT models, making it less straightforward to calculate sensitivity and bias.

Consider the recent study by Laskowska, Łęski, and Koziorowski (2015), in which patients with Parkinson’s disease, patients with schizophrenia, and healthy controls completed a test of facial emotion recognition. On each trial, subjects saw a photograph of a face and reported which of six possible emotions were conveyed, and which were not, by choosing one of three possibilities per emotion: “shown,” “not shown,” and “hard to say.” It was possible for a photograph to display more than one emotion. The authors analyzed the data using a standard SDT model by considering each response (six per photograph) as a separate YN trial. The authors found that, on average, the Parkinson’s disease and schizophrenia patients had lower sensitivity to facial emotions than the controls, and that the Parkinson’s disease patients adopted a more liberal response strategy than the controls. However, the authors had to recode “hard to say” responses before they could calculate sensitivity and bias, which vitiates the interpretation of their statistically significant group differences.

Other examples of experiments that do not conform to the requirements of regular SDT models can be found in the neuroimaging literature, where subjects are often required to perform specific tasks while their brain activity is recorded. Such experiments typically require strict control over the time intervals between trials and cannot guarantee that subjects will respond in time to every trial. In the functional magnetic resonance imaging (fMRI) study by Kreitewolf, Friederici, and von Kriegstein (2014), subjects performed two tasks. In one task, subjects heard sequences of words, and indicated whether each word was the same as or different to the word that preceded it. In the other task, subjects heard the same sequences, and indicated whether sequential words had the same or a different intonation (either rising or flat). The researchers were primarily interested in differences in brain activity related to phonological and lexical processing; therefore, to compensate for differences related to task difficulty, subjects' percent correct scores were used as covariates in the fMRI analysis. Missing responses were simply labelled as incorrect. Under these circumstances, SDT measures of sensitivity and bias might have been better covariates than percent correct.

In the present paper, we present a method for dealing with ambivalent or missing responses in YN and 2AFC experiments. This method involves modifying regular SDT models, and re-deriving the equations for estimating sensitivity and bias. We show that, for both YN and 2AFC experiments, the equations for estimating sensitivity are actually the same as those of the regular SDT models, but the equations for estimating bias are different. Furthermore, the modified SDT models allow psychologically meaningful measures that reflect the degree of uncertainty or ambivalence within the decision process to be estimated from the data. We performed simulations to test the utility of the modified models in practice, and examined how previously used recoding strategies (e.g., Kreitewolf et al., 2014; Laskowska et al., 2015) effect the estimates of SDT measures.

SDT models

Regular YN model

The *equal-variance Gaussian SDT model for YN experiments*, hereby referred to as the regular YN model, was the first SDT model to be described (Peterson, Birdsall, & Fox, 1954; Tanner & Swets, 1954) and forms the foundation of all SDT analysis. It has been outlined in detail by many previous authors (e.g., Green & Swets, 1966; Macmillan & Creelman, 2005). We outline it again briefly here, so that readers can see the difference between this model and the modified version, described later on.

Under the model, each trial of the experiment contains a single stimulus drawn at random from one of two classes, and the subject indicates to which class the stimulus belonged. Let X represent the stimulus class. When $X = 0$, the stimulus is drawn from the first class, and when $X = 1$, the stimulus is drawn from the second class. The subject then generates a random variable (or "observation") from the stimulus. These observations are denoted Ψ. If $X = 0$, Ψ has a standard normal probability distribution, $\Psi_{X=0} \sim \mathcal{N}(0, 1)$. If $X = 1$, Ψ has a normal probability distribution with mean d and unit standard deviation, $\Psi_{X=1} \sim \mathcal{N}(d, 1)$. When d is large, the two distributions are distinct, and the subject can easily distinguish between the stimulus classes. If $d = 0$, the classes are indistinguishable. Thus, d is a measure of sensitivity[1].

Let Y represent the subject's response to a trial. When $Y = 0$, the subject responds that the stimulus was in the first class, and when $Y = 1$, the subject responds that the stimulus was in the second class. Under the model, the subject makes a response based on whether Ψ is smaller or greater than some fixed threshold (or "criterion"), denoted k.

[1]In the literature, the more commonly used index of sensitivity is d', where the $'$ symbol represents the fact that the distance between the means of the two distributions of observations is "standardized" (i.e., d divided by the pooled standard deviation). However, since we assume here that the standard deviations of both distributions are 1, there is actually no difference between d and d'. The distinction between d and d' becomes important when either standard deviation is not set to 1, such as in unequal-variance SDT models.

Formally, this decision rule can be written

$$Y = 0 \text{ if } \Psi < k,$$

$$Y = 1 \text{ if } \Psi \geq k.$$

When $k = \frac{d}{2}$, the subject is said to be "unbiased," since both responses are equally likely. The distance between true k and unbiased k is denoted c. Thus, c is a measure of bias.

The model is summarized in the left panel of Fig. **??**. The figure shows two important additional variables. The blue shaded region represents the probability of a "false alarm," that is, when $Y = 1$ given $X = 0$. We denote this probability f. The green shaded region represents the probability of a "hit," that is, when $Y = 1$ given $X = 1$, denoted h. These conditional probabilities can be expressed in terms of d and c:

$$\begin{aligned} P\left\{Y = 1 \mid X = 0\right\} = f &= 1 - \Phi\left(k\right) \\ &= 1 - \Phi\left(c + \frac{d}{2}\right), \end{aligned}$$

where $\Phi\left(.\right)$ denotes the standard normal cumulative distribution function; and

$$\begin{aligned} P\left\{Y = 1 \mid X = 1\right\} = h &= 1 - \Phi\left(k - d\right) \\ &= 1 - \Phi\left(c - \frac{d}{2}\right). \end{aligned}$$

Crucially, because f and h can be estimated directly from the data in an experiment, the above equations can be combined and re-arranged to produce estimates of sensitivity and bias. Specifically, the equation for estimating sensitivity is

$$\hat{d} = \Phi^{-1}\left(\hat{h}\right) - \Phi^{-1}\left(\hat{f}\right), \tag{1}$$

where $\Phi^{-1}\left(.\right)$ denotes the probit function, and a caret denotes the *maximum-likelihood estimate* of a variable. The equation for estimating bias is

$$\hat{c} = -\frac{1}{2}\left[\Phi^{-1}\left(\hat{h}\right) + \Phi^{-1}\left(\hat{f}\right)\right]. \tag{2}$$

Verification of the model via Python code is provided in Appendix A.

It is worth noting that the assumptions of the regular YN model can be relaxed in various ways. For example, it is possible for the probability distributions of $\Psi_{X=0}$ and $\Psi_{X=1}$ to be non-Gaussian, or have different variances. Indeed, work on recognition memory has consistently reported phenomena that are inconsistent with the regular YN model but can be explained by assuming unequal variances (Wixted, 2007; Yonelinas & Parks, 2007) or by assuming that $\Psi_{X=0}$ and $\Psi_{X=1}$ are drawn from mixtures of latent distributions (DeCarlo, 2002). Another possibility is to allow k to vary across trials rather than remaining constant over trials (Cabrera, Lu, & Dosher, 2015). However, relaxing any of these assumptions means that the YN model is no longer exactly identified (i.e., the number of independent unknown variables and the number of independent known variables are not the same), so consequently cannot be solved for d and c. For the purposes of this article, we adhere to these assumptions, but note that future work could extend our models to incorporate unequal variance, non-Gaussian decision variables, and variable criteria.

Modified YN model

One can account for ambivalent or missing responses in YN experiments with two modifications to the regular YN model, as illustrated in the right panel of Fig. **??**. The first is that the modified model contains two criteria, a and b, rather than a single criterion. The second modification is to the decision rule. Let Z denote the subject's response. When $Z = 0$, the subject responds that the stimulus is in the first class. When $Z = 1$, the subject responds ambivalently ("I don't know"), or does not respond at all. When $Z = 2$, the subject responds that the stimulus is in the second class. This decision rule can be written as

$$Z = 0 \text{ if } \Psi < a,$$

$$Z = 1 \text{ if } a \leq \Psi < b,$$

$$Z = 2 \text{ if } \Psi \geq b.$$

We can define false-alarm and hit probabilities for this model in an analogous way to the regular model:

$$\begin{aligned} P\left\{Z = 2 \mid X = 0\right\} &= f = 1 - \Phi\left(b\right) \\ P\left\{Z = 2 \mid X = 1\right\} &= h = 1 - \Phi\left(b - d\right). \end{aligned}$$

By combining these equations and solving for d, and replacing f and h with their respective maximum-likelihood estimates from the data, we again arrive at

$$\hat{d} = \Phi^{-1}\left(\hat{h}\right) - \Phi^{-1}\left(\hat{f}\right).$$

In other words, the equation for estimating sensitivity under the modified YN model is the same as under the regular YN model (Eq. 1). No recoding of trials is necessary.

Under the modified YN model, a subject could be said to be unbiased if the criteria a and b were centered on $\frac{d}{2}$, since both affirmative responses would be equally likely in this case. Thus, an appropriate measure of bias, analogous to c from the regular YN model, is the distance between $\frac{d}{2}$ and the midpoint between a and b. We denote this quantity λ. It is related to a, b, and d via the equation

$$\lambda = \frac{1}{2}\left(a + b - d\right).$$

An additional probability is required to calculate λ. The red shaded region in the left panel of Fig. 1 shows $P\left\{Z = 1 \mid X = 0\right\}$, which we denote g for convenience. This probability can be expressed in terms of a and b:

$$P\left\{Z = 1 \mid X = 0\right\} = g = \Phi\left(b\right) - \Phi\left(a\right).$$

Like f and h, g can be estimated directly from the data—it is proportion of ambivalent or missing responses when the stimulus was drawn from the first class. Therefore, the above equations can be combined and rearranged to arrive at the equation for the maximum-likelihood estimate of λ:

$$\hat{\lambda} = -\frac{1}{2}\left[\Phi^{-1}\left(\hat{f} + \hat{g}\right) + \Phi^{-1}\left(\hat{h}\right)\right]. \tag{3}$$

The model allows a third psychologically meaningful measure to be calculated from the data. The distance between the two criteria, denoted u, represents the degree of uncertainty or ambivalence within the decision process. This measure can be estimated from the data using the equation

$$\hat{u} \ = \ \Phi^{-1}\left(\hat{f} + \hat{g}\right) - \Phi^{-1}\left(\hat{f}\right). \tag{4}$$

Note that, in the special case where $u = 0$, the modified model reduces to the regular YN model: λ is equivalent to c, and can be calculated using Eq. 2, as usual. Verification of the model via Python code is provided in Appendix B.

Regular 2AFC model

In YN experiments, subjects are presented with one stimulus per trial. In 2AFC experiments, subjects are presented with two stimuli per trial, one belonging to each stimulus class, usually in sequential order. SDT can be readily extended to such experiments, allowing researchers to calculate measures of sensitivity and bias (e.g., DeCarlo, 2012). In both YN and 2AFC models, "sensitivity" has the same meaning—it refers to the distance between the means of the stimulus classes. However, "bias" has a different meaning. In YN experiments, bias refers to a preference for selecting one stimulus class over the other; in 2AFC experiments, bias refers to a preference for one stimulus over the other (e.g., the first).

Let W denote the presentation order of the stimuli in a 2AFC experiment. When $W = 0$, the first stimulus is drawn from the first class, and the second stimulus is drawn from the second class. When $W = 1$, the first stimulus is drawn from the second class, and the second stimulus is drawn from the first class. The subject makes two observations per trial, Ψ_0 and Ψ_1, corresponding to the first and second stimuli, respectively, and then chooses whichever stimulus had the larger corresponding observation. In order to account for bias, a quantity is always added to the first observation prior to the decision being made (see DeCarlo, 2012). Let Y represent the subject's response to a trial. When $Y = 0$, the subject chooses the first stimulus, and when $Y = 1$, the subject chooses the second stimulus. This decision rule can be expressed formally as

$$Y = 0 \text{ if } \Psi_0 + l > \Psi_1,$$

$$Y = 1 \text{ if } \Psi_0 + l \leq \Psi_1,$$

where l is a measure of bias. The model is summarized in the top panels of Fig. 2. The top-left and top-right panels are illustrations of two different trials. In both of them, the subject has just made an observation of the first stimulus. The left panel shows a trial where this stimulus was drawn from the first class (i.e., $W = 0$). The blue shaded region in this figure represents the conditional probability $P\left\{Y = 1 \mid W = 0\right\}$, which for convenience we arbitrarily label as a false alarm, f. The panel makes it clear that, unlike under the YN model, the probability of making a false alarm varies from trial to trial depending on the value of Ψ_0. Specifically,

$$P\left\{Y = 1 \mid W = 0, \Psi_0 = x\right\} = 1 - \Phi\left(x + l\right)$$

where $x \sim \mathcal{N}\left(d, 1\right)$. The corresponding equation not conditional on x is

$$P\left\{Y = 1 \mid W = 0\right\} = f = \int \left[1 - \Phi\left(x + l\right)\right] \phi\left(x - d\right) \mathrm{d}x,$$

where $\phi\left(\cdot\right)$ is the normal probability density function. Conveniently, equations of this kind can be simplified, leading to

$$f = \Phi\left(\frac{-d - l}{\sqrt{2}}\right).$$

The top-right panel of Fig. 2 shows a trial where the first stimulus was drawn from the second class (i.e., $W = 1$). The green shaded region, $P\left\{Y = 1 \mid W = 1\right\}$ or h, is given by the following equations:

$$P\left\{Y = 1 \mid W = 1, \Psi_0 = y\right\} = 1 - \Phi\left(y + l - d\right),$$

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

$$\begin{aligned} P\{Y=1 \mid W=1\} = h &= \int [1-\Phi(y+l-d)]\,\phi(y)\,\mathrm{d}y \\ &= \Phi\left(\frac{d-l}{\sqrt{2}}\right). \end{aligned}$$

The maximum-likelihood estimate of d under the regular 2AFC model is therefore given by

$$\hat{d} = \frac{1}{\sqrt{2}}\left[\Phi^{-1}\left(\hat{h}\right) - \Phi^{-1}\left(\hat{f}\right)\right], \tag{5}$$

and the maximum-likelihood estimate of l is given by[2]

$$\hat{l} = -\frac{1}{\sqrt{2}}\left[\Phi^{-1}\left(\hat{h}\right) + \Phi^{-1}\left(\hat{f}\right)\right]. \tag{6}$$

Verification of the model via Python code is provided in Appendix C.

Modified 2AFC model

Ambivalent or missing responses in 2AFC experiments can be accomodated using a "yardstick" heuristic (DeCarlo, 2013). As under the regular 2AFC model, the subject makes two observations per trial, and chooses whichever stimulus has the largest corresponding observation (after taking into account bias). However, under the modified model, the subject also places a yardstick around Ψ_0, and makes an affirmative response only if Ψ_1 falls beyond the endpoints of the yardstick. Let Z denote the subject's response. When $Z=0$, the subject chooses the first stimulus. When $Z=1$, the subject responds ambivalently ("I don't know"), or does not respond at all. When $Z=2$, the subject chooses the second stimulus. The decision rule is

$$Z = 0 \text{ if } \quad \Psi_0 + \alpha > \Psi_1,$$

[2]Equation 6 does not appear in previous introductions to SDT, despite entire chapters of authoritative textbooks being devoted to 2AFC (e.g., Green & Swets, 1966; Macmillan & Creelman, 2005). In fact, Macmillan and Creelman suggest (p. 170), "For measuring response bias [in 2AFC], the methods [for YN] are entirely adequate. No $\sqrt{2}$ adjustment is necessary, because (as the reader may not be surprised to learn) bias in one task cannot be predicted from bias in the other."

$$\begin{aligned} Z = 1 \text{ if } & \begin{cases} \Psi_0 + \alpha \leq \Psi_1 \\ \text{and} \\ \Psi_0 + \beta > \Psi_1 \end{cases} \\ Z = 2 \text{ if } & \quad \Psi_0 + \beta \leq \Psi_1. \end{aligned}$$

Where α and β are the endpoints of the yardstick. This idea has been used in previous formulations of SDT models for same-different experiments (DeCarlo, 2013). The model is shown in the lower panels of Fig. 2. The bottom-left panel shows a trial where $W = 0$. The blue shaded region f is given by

$$P\{Z = 2 \mid W = 0\} = f = \Phi\left(\frac{-d - \beta}{\sqrt{2}}\right).$$

The bottom-right panel shows a trial where $W = 1$. The green shaded region h is given by

$$P\{Z = 2 \mid W = 1\} = h = \Phi\left(\frac{d - \beta}{\sqrt{2}}\right).$$

Combining and rearranging these equations reveals that the maximum-likelihood estimate of d under the modified 2AFC model is given by the same equation as under the regular 2AFC model (Eq. 5).

Under the modified 2AFC model, the subject could be said to be unbiased if the middle of the yardstick was always equal to the value of first observation. Therefore, an appropriate measure of bias is the distance from x, where x is an instance of Ψ_0, to the middle of the yardstick, or

$$\zeta = \frac{1}{2}(\beta - \alpha).$$

Since β is just l from the regular 2AFC model, it can be expressed uniquely in terms of h and f:

$$\beta = -\frac{1}{\sqrt{2}}\left[\Phi^{-1}(h) + \Phi^{-1}(f)\right].$$

By contrast, α requires an additional conditional probability, $P\{Z = 1 \mid W = 0\}$ (denoted g for convenience) which is given by

$$P\{Z = 1 \mid W = 0, \Psi_0 = x\} \quad = \quad \Phi(x + \beta) - \Phi(x - \alpha)$$

$$\begin{aligned} P\left\{Z=1 \mid W=0\right\}=g &= \int\left[\Phi\left(x+\beta\right)-\Phi\left(x-\alpha\right)\right]\phi\left(x\right)\mathrm{d}x \\ &= \Phi\left(\frac{\beta}{\sqrt{2}}\right)-\Phi\left(\frac{-\alpha}{\sqrt{2}}\right). \end{aligned}$$

Combining the above equations, re-arranging, and simplifying yields the maximum-likelihood estimator for ζ:

$$\hat{\zeta} = -\frac{1}{\sqrt{2}}\left[\Phi^{-1}\left(\hat{f}+\hat{g}\right)+\Phi^{-1}\left(\hat{h}\right)\right]. \tag{7}$$

Finally, an appropriate measure of uncertainty or ambivalence under this model is the width of the yardstick, denoted τ, which is simply $\alpha+\beta$. The equation that yields its maximum-likelihood estimate is

$$\hat{\tau} = \sqrt{2}\left[\Phi^{-1}\left(\hat{f}+\hat{g}\right)-\Phi^{-1}\left(\hat{f}\right)\right]. \tag{8}$$

Verification of the model via Python code is provided in Appendix D.

Simulations

In this section, we compare the efficiency of the modified models. By “efficiency,” we mean the ability to recover the models original parameters from simulated data sets.

Appendix A

Regular YN model

```
"""simulate_yn_regular.py"""
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt



def decide(psis, k):
        """The decision rule applied to an
        array of observations."""
        for psi in psis:
            if psi < k:
                yield 0
            else:
                yield 1



def sim_regular_yn(d, c, n0, n1=None):
    """Simulate data under the modified YN model.

    Parameters
    ----------
    d : float
        Measure of sensitivity.
    c : float
        Measure of bias.
```

```
n0 : int
    Number of trials with stimuli from the first
    class.
n1: int, optional
    Number of trials with stimuli from the second
    class; defaults to n1.

Returns
-------
f : int
    Count of observed false alarms.
h : int
    Count of observed hits.
g : int
    Count of observed type 1 ambivalent
    responses.
j : int
    Count of observed type 2 ambivalent
    responses.
m : int
    Count of observed misses.
r : int
    Count of observed correct rejections.
"""
if n1 is None:
    n1 = n0
k = d/2. + c
```

```
    psi_0 = norm.rvs(0, 1, n0)
    rsp_0 = np.array([x for x in decide(psi_0, k)])
    r, f = [sum(rsp_0 == i) for i in xrange(2)]
    psi_1 = norm.rvs(d, 1, n1)
    rsp_1 = np.array([x for x in decide(psi_1, k)])
    m, h = [sum(rsp_1 == i) for i in xrange(2)]
    return f, h, m, r


def est_regular_yn(f, h, m, r):
    """Calculate maximum-likelihood estimates of sens-
    itivity and bias.

    Parameters
    ----------
    f : int
        Count of observed false alarms.
    h : int
        Count of observed hits.
    m : int
        Count of observed misses.
    r : int
        Count of observed correct rejections.

    Returns
    -------
    d : float
```

```
        Measure of sensitivity.
    c : float
        Measure of bias.
    n0 : int
        Number of trials with stimuli from the first
        class.
    n1: int
        Number of trials with stimuli from the second
        class.
    """
    n0, n1 = float(f + r), float(h + m)
    if f == 0:
        f += 0.5
    if f == (f + r):
        f -= 0.5
    if h == 0:
        h += 0.5
    if h == (h + m):
        h -= 0.5
    fhat = f / n0
    hhat = h / n1
    d = norm.ppf(hhat) - norm.ppf(fhat)
    c = -0.5 * (norm.ppf(hhat) + norm.ppf(fhat))
    return d, c, f + r, h + m


def test0():
    """Plots the estimates of d as a function of true
    d from a series of simulated data sets. These
    estimates are very close to the corresponding
    true values, indicating that the model is valid."""
    c = 0.1
    true_d_vals = np.linspace(-5, 5, 51)
    for i, n in enumerate((25, 100, 250, 500), 1):
        plt.subplot(2, 2, i)
        est_d_vals = []
        for d in true_d_vals:
            dhat, chat, _, _ = est_regular_yn(
                *sim_regular_yn(d, c, n)
            )
            est_d_vals.append(dhat)
        plt.plot(true_d_vals, est_d_vals, ’o’)
        plt.grid()
        plt.xlim(-5, 5)
        plt.ylim(-5, 5)
        plt.plot(true_d_vals, true_d_vals, ’k’)
        plt.title(’%i trials’ % (n * 2))
        plt.xlabel(’$d$’)
        plt.ylabel(’$\hat{d}$’)
    plt.show()


def test1():
    """Plots the estimates of lambda as a function of true
    lambda from a series of simulated data sets. These
    estimates are very close to the corresponding
    true values, indicating that the model is valid."""
    d = 1.5
    true_c_vals = np.linspace(-2, 2, 51)
    for i, n in enumerate((25, 100, 250, 500), 1):
        plt.subplot(2, 2, i)
        est_c_vals = []
        for c in true_c_vals:
            dhat, chat, _, _ = est_regular_yn(
                *sim_regular_yn(d, c, n)
            )
            est_c_vals.append(chat)
        plt.plot(true_c_vals, est_c_vals, 'o')
        plt.grid()
        plt.xlim(-2, 2)
        plt.ylim(-2, 2)
        plt.plot(true_c_vals, true_c_vals, 'k')
        plt.title('%i trials' % (n * 2))
        plt.xlabel(r'$lamda$')
        plt.ylabel(r'$\hat{lamda}$')
    plt.show()


if __name__ == '__main__':
    test0()
    test1()
```

Appendix B

Modified YN model

```
"""simulate_yn_modified.py"""
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt



def decide(psis, a, b):
        """The decision rule applied to an
        array of observations."""
        for psi in psis:
            if psi < a:
                yield 0
            elif a <= psi < b:
                yield 1
            else:
                yield 2



def sim_modified_yn(d, lam, u, n0, n1=None):
    """Simulate data under the modified YN model.

    Parameters
    ----------
    d : float
        Measure of sensitivity.
```

```
lam : float
    Measure of bias.
u : float
    Measure of uncertainty/ambivalence.
n0 : int
    Number of trials with stimuli from the first
    class.
n1: int, optional
    Number of trials with stimuli from the second
    class; defaults to n1.

Returns
-------
f : int
    Count of observed false alarms.
h : int
    Count of observed hits.
g : int
    Count of observed type 1 ambivalent
    responses.
j : int
    Count of observed type 2 ambivalent
    responses.
m : int
    Count of observed misses.
r : int
    Count of observed correct rejections.
    """
    if n1 is None:
        n1 = n0
    a = d/2. + lam - u / 2.
    b = d/2. + lam + u / 2.
    psi_0 = norm.rvs(0, 1, n0)
    rsp_0 = np.array([x for x in decide(psi_0, a, b)])
    r, g, f = [sum(rsp_0 == i) for i in xrange(3)]
    psi_1 = norm.rvs(d, 1, n1)
    rsp_1 = np.array([x for x in decide(psi_1, a, b)])
    m, j, h = [sum(rsp_1 == i) for i in xrange(3)]
    return f, h, g, j, m, r


def est_modified_yn(f, h, g, j, m, r):
    """Calculate maximum-likelihood estimates of sens-
    itivity and bias.

    Parameters
    ----------
    f : int
        Count of observed false alarms.
    h : int
        Count of observed hits.
    m : int
        Count of observed misses.
    r : int
```

```
        Count of observed correct rejections.

    Returns
    -------
    d : float
        Measure of sensitivity.
    lam : float
        Measure of bias.
    u : float
        Measure of uncertainty/ambivalence.
    n0 : int
        Number of trials with stimuli from the first
        class.
    n1: int
        Number of trials with stimuli from the second
        class.
    """
    n0, n1 = float(f + r + g), float(h + m + j)
    if f == 0:
        f += 0.5
    if f == (f + r + g):
        f -= 0.5
    if h == 0:
        h += 0.5
    if h == (h + m + j):
        h -= 0.5
    fhat = f / n0
    hhat = h / n1
    ghat = g / n0
    d = norm.ppf(hhat) - norm.ppf(fhat)
    lam = -0.5 * (norm.ppf(hhat) + norm.ppf(fhat + ghat))
    if g == 0:
        u = 0
    else:
        u = norm.ppf(fhat + ghat) - norm.ppf(fhat)
    return d, lam, u, f + r + g, h + m + j


def test0():
    """Plots the estimates of d as a function of true
    d from a series of simulated data sets. These
    estimates are very close to the corresponding
    true values, indicating that the model is valid."""
    c = 0.1
    u = 0.4
    true_d_vals = np.linspace(-5, 5, 51)
    for i, n in enumerate((25, 100, 250, 500), 1):
        plt.subplot(2, 2, i)
        est_d_vals = []
        for d in true_d_vals:
            dhat, chat, uhat, _, _ = est_modified_yn(
                *sim_modified_yn(d, c, u, n)
            )
            est_d_vals.append(dhat)
        plt.plot(true_d_vals, est_d_vals, 'o')
        plt.grid()
        plt.xlim(-5, 5)
        plt.ylim(-5, 5)
        plt.plot(true_d_vals, true_d_vals, 'k')
        plt.title('%i trials' % (n * 2))
        plt.xlabel('$d$')
        plt.ylabel('$\hat{d}$')
    plt.show()


def test1():
    """Plots the estimates of lambda as a function of true
    lambda from a series of simulated data sets. These
    estimates are very close to the corresponding
    true values, indicating that the model is valid."""
    d = 1.5
    u = 0.6
    true_lam_vals = np.linspace(-2, 2, 51)
    for i, n in enumerate((25, 100, 250, 500), 1):
        plt.subplot(2, 2, i)
        est_lam_vals = []
        for lam in true_lam_vals:
            dhat, chat, uhat, _, _ = est_modified_yn(
                *sim_modified_yn(d, lam, u, n)
            )
            est_lam_vals.append(chat)
        plt.plot(true_lam_vals, est_lam_vals, 'o')
        plt.grid()
        plt.xlim(-2, 2)
        plt.ylim(-2, 2)
        plt.plot(true_lam_vals, true_lam_vals, 'k')
        plt.title('%i trials' % (n * 2))
        plt.xlabel(r'$lamda$')
        plt.ylabel(r'$\hat{lamda}$')
    plt.show()



def test2():
    """Plots the estimates of u as a function of true
    u from a series of simulated data sets. These
    estimates are very close to the corresponding
    true values, indicating that the model is valid."""
    d = 1.5
    lam = 0.2
    true_u_vals = np.linspace(0, 3, 51)
    for i, n in enumerate((25, 100, 250, 500), 1):
        plt.subplot(2, 2, i)
        est_u_vals = []
        for u in true_u_vals:
            dhat, chat, uhat, _, _ = est_modified_yn(
                *sim_modified_yn(d, lam, u, n)
            )
            est_u_vals.append(uhat)
        plt.plot(true_u_vals, est_u_vals, ’o’)
        plt.grid()
        plt.xlim(0, 3)
        plt.ylim(0, 3)
        plt.plot(true_u_vals, true_u_vals, ’k’)
        plt.title(’%i trials’ % (n * 2))
        plt.xlabel(r’$u$’)
        plt.ylabel(r’$\hat{u}$’)
    plt.show()


if __name__ == ’__main__’:
    test0()
    test1()
    test2()
```

Appendix C

Regular 2AFC model

```
"""simulate_2afc_regular.py"""
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt



def decide(psi_pairs, l):
        """The decision rule applied to an
        array of observations."""
        for psi_0, psi_1 in psi_pairs:
            if psi_0 + l > psi_1:
                yield 0
            else:
                yield 1



def sim_regular_2afc(d, l, n0, n1=None):
    """Simulate data under the modified YN model.

    Parameters
    ----------
    d : float
        Measure of sensitivity.
    l : float
        Measure of bias.
    n0 : int
        Number of trials with stimuli from the first
        class.
    n1: int, optional
        Number of trials with stimuli from the second
        class; defaults to n1.

    Returns
    -------
    f : int
        Count of observed false alarms.
    h : int
        Count of observed hits.
    m : int
        Count of observed misses.
    r : int
        Count of observed correct rejections.
    """
    if n1 is None:
        n1 = n0
    psi_pairs_0 = zip(norm.rvs(0, 1, n0), norm.rvs(d, 1, n0))
    rsp_0 = np.array([x for x in decide(psi_pairs_0, l)])
    m, h = [sum(rsp_0 == i) for i in xrange(2)]
    psi_pairs_1 = zip(norm.rvs(d, 1, n1), norm.rvs(0, 1, n1))
    rsp_1 = np.array([x for x in decide(psi_pairs_1, l)])
    r, f = [sum(rsp_1 == i) for i in xrange(2)]
    return f, h, m, r
```

```
def est_regular_2afc(f, h, m, r):
    """Calculate maximum-likelihood estimates of sens-
    itivity and bias.

    Parameters
    ----------
    f : int
        Count of observed false alarms.
    h : int
        Count of observed hits.
    m : int
        Count of observed misses.
    r : int
        Count of observed correct rejections.

    Returns
    -------
    d : float
        Measure of sensitivity.
    c : float
        Measure of bias.
    n0 : int
        Number of trials with stimuli from the first
        class.
    n1: int
```

```
        Number of trials with stimuli from the second
        class.
    """
    n1, n0 = float(f + r), float(h + m)
    if f == 0:
        f += 0.5
    if f == (f + r):
        f -= 0.5
    if h == 0:
        h += 0.5
    if h == (h + m):
        h -= 0.5
    fhat = f / n1
    hhat = h / n0
    d = (norm.ppf(hhat) - norm.ppf(fhat))/np.sqrt(2)
    l = -(norm.ppf(hhat) + norm.ppf(fhat))/np.sqrt(2)
    return d, l, f + r, h + m


def test0():
    """Plots the estimates of d as a function of true
    d from a series of simulated data sets. These
    estimates are very close to the corresponding
    true values, indicating that the model is valid."""
    l = 0.1
    true_d_vals = np.linspace(-5, 5, 51)
    for i, n in enumerate((25, 100, 250, 500), 1):
        plt.subplot(2, 2, i)
        est_d_vals = []
        for d in true_d_vals:
            dhat, lhat, _, _ = est_regular_2afc(
                *sim_regular_2afc(d, l, n)
            )
            est_d_vals.append(dhat)
        plt.plot(true_d_vals, est_d_vals, 'o')
        plt.grid()
        plt.xlim(-5, 5)
        plt.ylim(-5, 5)
        plt.plot(true_d_vals, true_d_vals, 'k')
        plt.title('%i trials' % (n * 2))
        plt.xlabel('$d$')
        plt.ylabel('$\hat{d}$')
    plt.show()


def test1():
    """Plots the estimates of l as a function of true
    l from a series of simulated data sets. These
    estimates are very close to the corresponding
    true values, indicating that the model is valid."""
    d = 1.5
    true_l_vals = np.linspace(-2, 2, 51)
    for i, n in enumerate((25, 100, 250, 500), 1):
        plt.subplot(2, 2, i)
        est_l_vals = []
        for l in true_l_vals:
            dhat, lhat, _, _ = est_regular_2afc(
                *sim_regular_2afc(d, l, n)
            )
            est_l_vals.append(lhat)
        plt.plot(true_l_vals, est_l_vals, 'o')
        plt.grid()
        plt.xlim(-2, 2)
        plt.ylim(-2, 2)
        plt.plot(true_l_vals, true_l_vals, 'k')
        plt.title('%i trials' % (n * 2))
        plt.xlabel(r'$l$')
        plt.ylabel(r'$\hat{l}$')
    plt.show()


if __name__ == '__main__':
    test0()
    test1()
```

Appendix D

Modified 2AFC model

```
"""simulate_2afc_regular.py"""

import numpy as np

from scipy.stats import norm

import matplotlib.pyplot as plt




def decide(psi_pairs, alpha, beta):

        """The decision rule applied to an

        array of observations."""

        for psi_0, psi_1 in psi_pairs:

            if psi_0 + alpha > psi_1:

                yield 0

            elif psi_0 + alpha <= psi_1 and psi_0 + beta > psi_1:

                yield 1

            else:

                yield 2




def sim_modified_2afc(d, zeta, tau, n0, n1=None):

    """Simulate data under the modified YN model.


    Parameters

    ----------

    d : float

        Measure of sensitivity.
```

```
zeta : float
    Measure of bias.
tau : float
    Measure of uncertainty/ambivalence.
n0 : int
    Number of trials with stimuli from the first
    class.
n1: int, optional
    Number of trials with stimuli from the second
    class; defaults to n1.

Returns
-------
f : int
    Count of observed false alarms.
h : int
    Count of observed hits.
g : int
    Count of observed type 1 ambivalent
    responses.
j : int
    Count of observed type 2 ambivalent
    responses.
m : int
    Count of observed misses.
r : int
    Count of observed correct rejections.
    """
    if n1 is None:
        n1 = n0
    alpha = zeta - tau/2.
    beta = zeta + tau/2.
    psi_pairs_0 = zip(norm.rvs(0, 1, n0), norm.rvs(d, 1, n0))
    rsp_0 = np.array([x for x in decide(psi_pairs_0, alpha, beta)])
    m, j, h = [sum(rsp_0 == i) for i in xrange(3)]
    psi_pairs_1 = zip(norm.rvs(d, 1, n1), norm.rvs(0, 1, n1))
    rsp_1 = np.array([x for x in decide(psi_pairs_1, alpha, beta)])
    r, g, f = [sum(rsp_1 == i) for i in xrange(3)]
    return f, h, g, j, m, r


def est_modified_2afc(f, h, g, j, m, r):
    """Calculate maximum-likelihood estimates of sens-
    itivity and bias.

    Parameters
    ----------
    f : int
        Count of observed false alarms.
    h : int
        Count of observed hits.
    g : int
        Count of observed type 1 ambivalent
        responses.
```

```
j : int
    Count of observed type 2 ambivalent
    responses.
m : int
    Count of observed misses.
r : int
    Count of observed correct rejections.

Returns
-------
d : float
    Measure of sensitivity.
zeta : float
    Measure of bias.
tau:
    Measure of uncertainty/ambivalence.
n0 : int
    Number of trials with stimuli from the first
    class.
n1: int
    Number of trials with stimuli from the second
    class.
"""
n0, n1 = float(f + r + g), float(h + m + j)
if f == 0:
    f += 0.5
if f == (f + r + g):
        f -= 0.5
    if h == 0:
        h += 0.5
    if h == (h + m + j):
        h -= 0.5
    fhat = f / n0
    hhat = h / n1
    ghat = g / n0
    d = (norm.ppf(hhat) - norm.ppf(fhat))/np.sqrt(2)
    zeta = -(norm.ppf(hhat) + norm.ppf(fhat + ghat))/np.sqrt(2)
    if g == 0:
        tau = 0
    else:
        tau = (norm.ppf(fhat + ghat) - norm.ppf(fhat))*np.sqrt(2)
    return d, zeta, tau, f + r, h + m


def test0():
    """Plots the estimates of d as a function of true
    d from a series of simulated data sets. These
    estimates are very close to the corresponding
    true values, indicating that the model is valid."""
    zeta = 0.1
    tau = 0.4
    true_d_vals = np.linspace(-5/np.sqrt(2), 5/np.sqrt(2), 51)
    for i, n in enumerate((25, 100, 250, 500), 1):
        plt.subplot(2, 2, i)
        est_d_vals = []
        for d in true_d_vals:
            dhat, zhat, that, _, _ = est_modified_2afc(
                *sim_modified_2afc(d, zeta, tau, n)
            )
            est_d_vals.append(dhat)
        plt.plot(true_d_vals, est_d_vals, 'o')
        plt.grid()
        plt.xlim(-5/np.sqrt(2), 5/np.sqrt(2))
        plt.ylim(-5/np.sqrt(2), 5/np.sqrt(2))
        plt.plot(true_d_vals, true_d_vals, 'k')
        plt.title('%i trials' % (n * 2))
        plt.xlabel('$d$')
        plt.ylabel('$\hat{d}$')
    plt.show()


def test1():
    """Plots the estimates of l as a function of true
    l from a series of simulated data sets. These
    estimates are very close to the corresponding
    true values, indicating that the model is valid."""
    d = 0
    tau = 1
    true_vals = np.linspace(-2, 2, 51)
    for i, n in enumerate((25, 100, 250, 500), 1):
        plt.subplot(2, 2, i)
        est_vals = []
        for zeta in true_vals:
            dhat, zhat, that, _, _ = est_modified_2afc(
                *sim_modified_2afc(d, zeta, tau, n)
            )
            est_vals.append(zhat)
        plt.plot(true_vals, est_vals, 'o')
        plt.grid()
        plt.xlim(-2, 2)
        plt.ylim(-2, 2)
        plt.plot(true_vals, true_vals, 'k')
        plt.title('%i trials' % (n * 2))
        plt.xlabel(r'$zeta$')
        plt.ylabel(r'$\hat{zeta}$')
    plt.show()


def test2():
    """Plots the estimates of u as a function of true
    u from a series of simulated data sets. These
    estimates are very close to the corresponding
    true values, indicating that the model is valid."""
    d = 1.5
    zeta = 0.2
    true_t_vals = np.linspace(0, 3, 51)
    for i, n in enumerate((25, 100, 250, 500), 1):
        plt.subplot(2, 2, i)
        est_t_vals = []
        for tau in true_t_vals:
            dhat, zhat, that, _, _ = est_modified_2afc(
                *sim_modified_2afc(d, zeta, tau, n)
            )
            est_t_vals.append(that)
        plt.plot(true_t_vals, est_t_vals, 'o')
        plt.grid()
        plt.xlim(0, 3)
        plt.ylim(0, 3)
        plt.plot(true_t_vals, true_t_vals, 'k')
        plt.title('%i trials' % (n * 2))
        plt.xlabel(r'$tau$')
        plt.ylabel(r'$\hat{tau}$')
    plt.show()


if __name__ == '__main__':
    test0()
    test1()
    test2()
```