MANUAL
SenPy: Algorithms for Sensitivity Testing in Python

Alex Casey, David Arthur

Contents

1 Introduction

Installation

To jump straight to package installation instructions see section 3.

Sensitivity tests are commonly used to estimate hidden (or latent) distribution parameters. For example, upon producing a batch of explosives it is assumed that each explosive specimen has a critical threshold value that, upon surpassing, will result in a positive response (detonation, a 'go'). The critical threshold values are assumed to be a continuous variable with an unknown probability density function. The difficulty of this problem arises because the critical threshold of each specimen cannot be directly measured. Consider the case of determining the yield or fracture threshold of a given material. These thresholds can be directly measured by slowing increasing the stress (or rather force) applied to the specimen with a tensile testing machine and noting the stress at which yielding or fracture occurs. An analogous test does not exist when trying to determine the critical threshold values of explosives.

When testing the sensitivity of explosives to impact or shock, one can impose a non-varying stress level (input pressure, drop-weight height, etc.) on the test article (explosive specimen) and note whether a response (detonation, 'go' or 'no-go') occurs. Unlike the fracture case, the input level cannot be slowly augmented until detonation is achieved. Additionally, if a positive response is not achieved the specimen cannot simply be re-tested at a higher input level because the specimen may have been damaged by the original test. Damage (void creation, phase change, etc.) is theorized to alter the sensitivity, or critical threshold, of the specimen – which is obviously unwanted since this is exactly the value one wishes to measure.

Note, that while explosives testing is used as a running example, the sensitivity testing scenario occurs in many fields. Take for example the response of a patient after receiving a prescribed dosage of medication. The patient may exhibit a change, or not – a 'go' or a 'no-go'. If no change is observed, you cannot simply prescribe an increased dose of medication to the patient (at least without waiting a responsible amount of time) because the effect of the previous treatment may alter the outcome. Other examples include the testing of insecticides, germicides, psycho-physical research, etc.

Sensitivity tests are herein defined as methods to estimate the parameters which describe the probability density function of critical threshold values (or otherwise describe the stimulus-response function in the case of non-parametric methods); where these critical thresholds cannot be directly measured (hidden, latent).

Many methods have been produced to both analyze sensitivity test data and to suggest new stimulus levels for subsequent testing. The latter is a procedure referred to as "sequential design." A few of these methods are (or will be made available) in SenPy. For each method, this manual contains a section on its background, theory and mathematical framework, and how to implement it in SenPy.

Methods currently available are listed here in black. Methods in development are listed in red.

- Bruceton (up-down) method
- Neyer
- Dror-Steinberg (a Bayesian approach)
- Isotonic regression
- Gaussian process classifier (with monotonicity constraint)

2 Methods

2.1 Bruceton (up-down)

IN DEVELOPMENT

2.1.1 Background

2.1.2 Model Overview

2.1.3 SenPy Implementation

2.2 Neyer

2.2.1 Background

In 1989, Barry T. Neyer published a paper outlining a 'new sensitivity' test and detailed its advantages over previous methods such a Probit, Bruceton, Robbins-Monro, and Langlie [1]. Later, in 1994, Neyer published an updated version of his sensitivity test; altering the sequential design algorithm (described later) [2]. The test assumes that the distribution of critical thresholds is modelled by a normal or logistic probability density function (pdf), or can be easily transformed to make the distribution normal. Normal and logistic distributions are parameterized by two variables, μ and σ. Neyer estimates these parameters using maximum likelihood estimators. As a point of notation, $\hat{\mu}$ and $\hat{\sigma}$ are the *estimates* or *estimators* of the true parameters μ and σ. An in-depth description of the maximum likelihood estimation to sensitivity tests is attributed to Cornfiled and Mantel [3] and Golub and Grubbs [4] - with the method originating in earlier publications by Dixon and Mood [5] and by Finney [6], Bliss [7], and Fisher [8].

Once the maximum likelihood estimates (MLEs) have been calculated, Neyer proposed a sequential design in which the next stimulus level to be tested is that which maximizes the determinate of the Fisher information matrix [2]. This essentially attempts to simultaneously minimize the variances of the estimated parameters, $\hat{\mu}$ and $\hat{\sigma}$, by decreasing the Cramér-Rao lower bound (CRLB) of said parameters. Although, as will be described later, the MLE of σ is not unbiased - and therefore is not bounded by the CRLB - minimizing the CRLB will generally decrease the variance of $\hat{\sigma}$ because they are computed from the same likelihood function.

Unique maximum likelihood estimates will not be obtained unless the stimulus level of the smallest positive response is less than the stimulus level of the greatest non-response (no "overlap" in responses) [9]. That is, the MLE will return a value of zero for $\hat{\sigma}$ and any value between the greatest non-response level and smallest positive response level for $\hat{\mu}$. Neyer's sequential design algorithm overcomes this deficiency by using an expandable binary-search algorithm to test points until overlap is achieved.

While much of the information outlined in Neyer's papers - in terms of the statistics - was previously published, Neyer greatly impacted the field by bringing the test to the energetic materials community, comparing the test to all other known methods, creating a robust sequential design algorithm, and producing a commercial software implementation of the test.

Specific descriptions of the maximum likelihood estimators, the information matrix, and the sequential design algorithm are given below.

2.2.2 Model Overview

Using "plate" notation, the current problem framework is shown graphically in figure 1. In this figure i is the specimen number and there are n total specimens to be tested. x_i is the stimulus level at which specimen i is tested and y_i is the specimen response (binary outcome). h_i is the actual threshold of specimen i (hidden variable), and all of the specimens are assumed to have threshold values that originate from a distribution uniquely determined by two parameters, μ and σ.

Under this model the probability y_i is easily described as

$$p(y_i = 1|x_i, h_i) = \begin{cases} 1, & \text{if } x_i \geq h_i \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

or

$$p(y_i = 0|x_i, h_i) = \begin{cases} 1, & \text{if } x_i < h_i \\ 0, & \text{otherwise .} \end{cases} \tag{2}$$

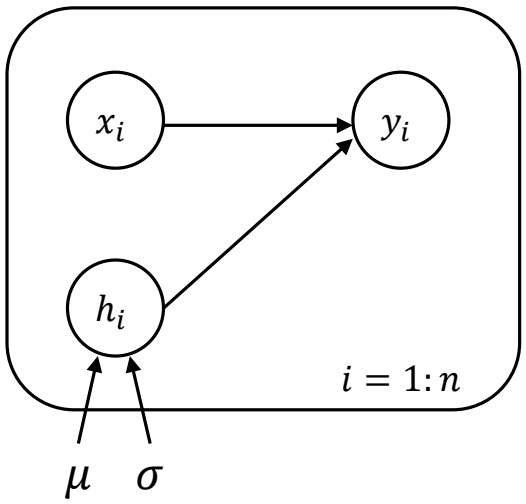


Figure 1: Graph demonstrating interaction of sensitivity model random variables assuming that the threshold distribution is parameterized by two variables, μ and σ.

Now, maintaining generality, it's assumed that the distribution h_i is given by a set of parameters $\tilde{\theta}$, which is written as $p(h_i|\tilde{\theta})$. Following the rules of probability, the joint distribution of y_i and h_i is

$$p(y_i, h_i|x_i, \vec{\theta}) = p(y_i|x_i, h_i)p(h_i|\vec{\theta}). \tag{3}$$

h_i can be marginalized out to produce a probability measure for y_i that is given given x_i and $\tilde{\theta}$ as

$$p(y_i|x_i, \vec{\theta}) = \int_{-\infty}^{\infty} p(y_i|x_i, h_i)p(h_i|\vec{\theta})dh_i. \tag{4}$$

Now, a 'nested' description of $p(y_i|x_i, h_i)$ could be produced, but it is easier to work with the case of $p(y_i = 1|x_i, h_i)$ which was intuitively formed and shown in equation 1. Making this substitution into equation 4 yields

$$p(y_i = 1|x_i, \vec{\theta}) = \int_{-\infty}^{\infty} p(y_i = 1|x_i, h_i)p(h_i|\vec{\theta})dh_i. \tag{5}$$

According to equation 1, $p(y_i = 1)$ is equal to one when $x_i \geq h_i$ and is zero otherwise. Thus, in the integral of equation 5 the integrand is zero for all values $h_i > x_i$ so that x_i can replace ∞ as the integral upper limit.

$$p(y_i = 1|x_i, \vec{\theta}) = \int_{-\infty}^{x_i} (1)p(h_i|\vec{\theta})dh_i. \tag{6}$$

Once $p(y_i = 1)$ is known, $p(y_i = 0)$ is simply given by $1 - p(y_i = 1)$ and the likelihood function can be formed. In Neyer and preceeding works, it is assumed that h_i is described by either a normal or logistic distribution parameterized by μ and σ. If the the normal distribution assumption is made then $p(h_i) \sim N(\mu, \sigma)$ and equation 6 becomes

$$p(y_i = 1|x_i, \vec{\theta}) = \int_{-\infty}^{x_i} N(\mu, \sigma)dh_i = \Phi(x_i|\mu, \sigma) \tag{7}$$

where $\Phi(x_i|\mu, \sigma)$ is the cumulative distribution function (cdf) of a normal distribution with given parameters μ and σ. Likewise, if a logistic distribution assumption is made then equation 6 reduces to

$$p(y_i = 1|x_i, \vec{\theta}) = \int_{-\infty}^{x_i} Logistic(\mu, \sigma)dh_i = S(x_i|\mu, \sigma) \tag{8}$$

where $S(x_i|\mu, \sigma)$ is the cdf of the logistic distribution;

$$S(x_i|\mu, \sigma) = \frac{1}{1 + exp\left(\frac{-(x_i - \mu)}{\sigma}\right)}. \tag{9}$$

Derivation: Maximum Likelihood Estimators
Here, we will used the notation used by Neyer and the previous authors.

Let $L^{(i)}$ represent the stimulus level to be tested on specimen i. The standardized level is then given by $z^{(i)} = \frac{L^{(i)}-\mu}{\sigma}$, where μ and σ are the parameters that describe the assumed normal distribution of critical threshold values. Then the probability of observing a positive response in a randomly selected specimen is

$$p(z^{(i)}) = \int_{-\infty}^{z^{(i)}} f(t)dt \tag{10}$$

where,

$$f(t) = \frac{1}{\sqrt{2\pi}} e^{\frac{-t^2}{2}} \tag{11}$$

Let $y^{(i)}$ represent the observed response at stimulus level $L^{(i)}$; and let $y^{(i)} = 1$ and $y^{(i)} = 0$ for a positive and negative response ('go', 'no go'), respectively. Because the response is described by a binary variable, the likelihood function follows the form of a binomial model

$$L(\mu, \sigma) = \prod_{i=1}^{n} p(z^{(i)})^{y} (1 - p(z^{(i)}))^{1-y} \tag{12}$$

where n is the number of specimens tested. If the same stimulus level is tested more than once then let $N^{(i)}$ and $M^{(i)}$ represent the number of positive and negative responses observed at $L^{(i)}$, respectively. Then, the total number of tests performed at level $L^{(i)}$ is equal to $(N^{(i)} + M^{(i)})$. Using this notation, equation 12 can be written as

$$L(\mu, \sigma) = \prod_{i=1}^{n} \binom{N^{(i)} + M^{(i)}}{N^{(i)}} p(z^{(i)})^{N^{(i)}} (1 - p(z^{(i)}))^{M^{(i)}} \tag{13}$$

Additionally, $(1 - p(z^{(i)}))$ is equivalent to $p(-z^{(i)})$ and can be referred to as $q^{(i)}$ so that the likelihood function is

$$L(\mu, \sigma) = \prod_{i=1}^{n} \binom{N^{(i)} + M^{(i)}}{N^{(i)}} p(z^{(i)})^{N^{(i)}} q(z^{(i)})^{M^{(i)}} \tag{14}$$

The values of μ and σ that maximize the likelihood function for the data set $\{(L^{(1)}, N^{(1)}, M^{(1)}),\ (L^{(2)}, N^{(2)}, M^{(2)}),\ \ldots\ (L^{(n)}, N^{(n)}, M^{(n)})\}$ are the MLEs $\hat{\mu}$ and $\hat{\sigma}$. Note that in the notation of equation 14, n now represents the number of unique stimulus levels tested and not the total number of tests performed.

The parameters that maximize the likelihood function are the same as those that maximize the natural logarithm of the likelihood function because the natural logarithm is a monotonically increasing function. The log-likelihood function is often easier to work with analytically, and additionally, it is the log-likelihood function that is used in the definition of the Fisher information matrix used later. Thus, the log-likelihood function is

$$l = log(L) = \sum_{i-1}^{n} \left[log\binom{N^{(i)} + M^{(i)}}{N^{(i)}} + N^{(i)} log\left(p(z^{(i)})\right) + M^{(i)} log\left(q(z^{(i)})\right) \right]. \tag{15}$$

The first order partial derivatives of equation 15 are useful to compute the MLE, as all of the first order partial derivatives must equal zero when the log-likelihood is maximized. These are derived as follows:

$$\frac{\partial l}{\partial \mu} = \sum_{i=1}^{n} \left[N^{(i)} \frac{\partial \left(log(p(z^{(i)}))\right)}{\partial p(z^{(i)})} \frac{\partial p(z^{(i)})}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial \mu} + M^{(i)} \frac{\partial \left(log(q(z^{(i)}))\right)}{\partial q(z^{(i)})} \frac{\partial q(z^{(i)})}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial \mu} \right] \tag{16}$$

$$\frac{\partial l}{\partial \sigma} = \sum_{i=1}^{n} \left[N^{(i)} \frac{\partial \left(log(p(z^{(i)}))\right)}{\partial p(z^{(i)})} \frac{\partial p(z^{(i)})}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial \sigma} + M^{(i)} \frac{\partial \left(log(q(z^{(i)}))\right)}{\partial q(z^{(i)})} \frac{\partial q(z^{(i)})}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial \sigma} \right] \tag{17}$$

As seen, equations 16 and 17 are derived by the continuous, albeit tedious, application of the "chain rule" in calculus. The partial derivatives found in these equations are formulated as

$$\frac{\partial \left(log(p(z^{(i)}))\right)}{\partial p(z^{(i)})} = \frac{1}{p(z^{(i)})} \tag{18}$$

$$\frac{\partial \left(log(q(z^{(i)}))\right)}{\partial q(z^{(i)})} = \frac{1}{q(z^{(i)})} \tag{19}$$

$$\frac{\partial p(z^{(i)})}{\partial z^{(i)}} = f(z^{(i)}) \tag{20}$$

$$\frac{\partial (z^{(i)})}{\partial z^{(i)}} = -f(z^{(i)}) \tag{21}$$

$$\frac{\partial z^{(i)}}{\partial \mu} = \frac{-1}{\sigma} \tag{22}$$

$$\frac{\partial z^{(i)}}{\partial \sigma} = \frac{-(L^{(i)} - \mu)}{\sigma^2} \tag{23}$$

Substituting equations 18 – 23 into equations 16 and 17 yields

$$\frac{\partial l}{\partial \mu} = \sum_{i=1}^{n} \left[N^{(i)} \frac{1}{p(z^{(i)})} f(z^{(i)}) \frac{-1}{\sigma} + M^{(i)} \frac{1}{q(z^{(i)})} f(z^{(i)}) \frac{1}{\sigma} \right] \tag{24}$$

$$\frac{\partial l}{\partial \sigma} = \sum_{i=1}^{n} \left[N^{(i)} \frac{1}{p(z^{(i)})} f(z^{(i)}) \frac{-(L^{(i)} - \mu)}{\sigma^2} + M^{(i)} \frac{1}{q(z^{(i)})} f(z^{(i)}) \frac{L^{(i)} - \mu}{\sigma^2} \right] \tag{25}$$

These derivatives are useful as they will almost certainly be used in any optimization routine or algorithm in order to calculate the parameter values which maximize the log-likelihood.

Derivation: Information Matrix

The Fisher information matrix is defined as

$$INF = -E\left[H[log(L)]\right] \tag{26}$$

where E$[\cdot]$ is the expectation operator, H$[\cdot]$ is the Hessian, and L is the likelihood function. The log-likelihood function was previously defined as *l* and the Hessian is the matrix of second partial derivatives. The partial derivatives are taken with respect to the parameters being estimated; in this case, μ and σ. Therefore,

$$H\left[log(L)\right] = H\left[l\right] = \begin{bmatrix} \frac{\partial^2 l}{\partial \mu^2} & \frac{\partial^2 l}{\partial \sigma \partial \mu} \\ \frac{\partial^2 l}{\partial \mu \partial \sigma} & \frac{\partial^2 l}{\partial \sigma^2} \end{bmatrix} \tag{27}$$

Let the elements of the information matrix be described as

$$INF = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \tag{28}$$

and the inverse of the information matrix be

$$[INF]^{-1} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \tag{29}$$

Under this description b_{11} and b_{22} are the Cramér-Rao lower bounds (CRLBs) for $\hat{\mu}$ and $\hat{\sigma}$, respectively. However, the CRLB only applies to estimators that are unbiased. If $\hat{\mu}$ is the estimator for μ and $\hat{\sigma}$ for σ, then

$\hat{\mu}$ and $\hat{\sigma}$ are unbiased estimators of μ and σ if $\mathrm{E}[\hat{\mu}] = \mu$ and $\mathrm{E}[\hat{\sigma}] = \sigma$. Rather than attempt to solve for $\hat{\mu}$ and $\hat{\sigma}$ when equations 24 and 25 are equal to zero and then take the expectation of those expressions, a simple simulation is run with synthetic data.

Consider the latent random variable, which represents the critical threshold values, to be given by a normal distribution $\sim \mathrm{N}(\mu, \sigma)$. For the simulation, let $\mu = 40$ and $\sigma = 3$. n stimulus levels are tested; stimulus levels are spaced equidistant from each other between the range of $[41, 49]$. The outcome of each test is decided randomly via a Monte-Carlo simulation. For every value of n the MLE of $\hat{\mu}$ and $\hat{\sigma}$ is calculated and this simulation is repeated 2000 times (holding n constant); the values of $\hat{\mu}$ and $\hat{\sigma}$ are averaged together in order to estimate $\mathrm{E}[\hat{\mu}]$ and $\mathrm{E}[\hat{\sigma}]$. The results for $\mathrm{n} = 3$ to 39 are shown in figure 2.2.2. This figure demonstrates that $\hat{\mu}$ is an unbiased estimator but $\hat{\sigma}$ is a biased estimator; although the bias decreases with increasing sample size n. Neyer estimated the bias of $\hat{\sigma}$ over a wide variety of designs [2] (permutations of good and poor initial guesses of the parameters) and approximated the relative bias as

$$relative\ bias = \frac{-3.5}{n}. \tag{30}$$

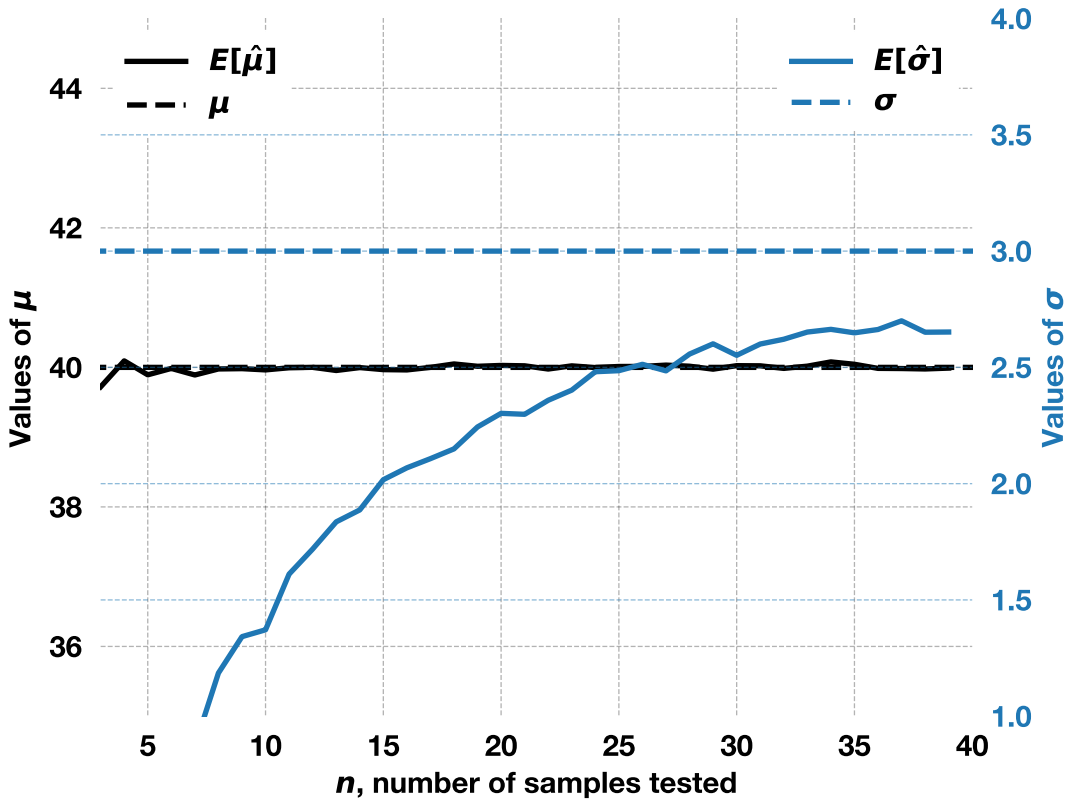


Figure 2: Expected values of estimators $\hat{\sigma}$ and $\hat{\sigma}$ for simulated data

However, despite the biasedness of $\hat{\sigma}$, minimizing the CRLB will decrease the variance of $\hat{\sigma}$ since both the CRLB and $\hat{\sigma}$ are derived from the same likelihood function.

Moving forward, minimizing the variance of $\hat{\mu}$ and $\hat{\sigma}$, or decreasing the values of b_{11} and b_{22}, is related to maximizing the values of a_{11} and a_{22} in the information matrix (*see last paragraph of this section*). Referring back to equations 26 and 27, it can be seen that these terms are

$$a_{11} = -E\left[\frac{\partial^2 l}{\partial \mu^2}\right] \tag{31}$$

$$a_{22} = -E\left[\frac{\partial^2 l}{\partial \sigma^2}\right] \tag{32}$$

recalling that *l* is the log-likelihood function given by equation 15. A derivation of the a_{11} term is provided here starting from equation 24. The end result of the a_{12} and a_{22} terms are given but an explicit derivation is not provided as these follow a similar form to the derivation of a_{11}.

$$\frac{\partial^2 l}{\partial \mu^2} = \sum_{i=1}^{n}\left[N^{(i)}\frac{-1}{\sigma}\frac{\partial\left(\frac{f(z^{(i)})}{p(z^{(i)})}\right)}{\partial \mu} + M^{(i)}\frac{-1}{\sigma}\frac{\partial\left(\frac{-f(z^{(i)})}{q(z^{(i)})}\right)}{\partial \mu}\right] \tag{33}$$

$$\frac{\partial\left(\frac{f(z^{(i)})}{p(z^{(i)})}\right)}{\partial \mu} = \frac{1}{p(z^{(i)})}\frac{\partial\left(f(z^{(i)})\right)}{\partial \mu} + f(z^{(i)})\frac{-1}{\left(p(z^{(i)})\right)^2}\frac{\partial\left(p(z^{(i)})\right)}{\partial \mu} \tag{34}$$

$$\frac{\partial\left(\frac{-f(z^{(i)})}{q(z^{(i)})}\right)}{\partial \mu} = \frac{-1}{q(z^{(i)})}\frac{\partial\left(f(z^{(i)})\right)}{\partial \mu} + f(z^{(i)})\frac{1}{\left(q(z^{(i)})\right)^2}\frac{\partial\left(q(z^{(i)})\right)}{\partial \mu} \tag{35}$$

$$\frac{\partial\left(f(z^{(i)})\right)}{\partial \mu} = \frac{\partial f(z^{(i)})}{\partial z^{(i)}}\frac{\partial z^{(i)}}{\partial \mu} = z^{(i)}f(z^{(i)})\frac{1}{\sigma} \tag{36}$$

$$\frac{\partial\left(p(z^{(i)})\right)}{\partial \mu} = \frac{\partial\left(p(z^{(i)})\right)}{\partial z^{(i)}}\frac{\partial z^{(i)}}{\partial \mu} = f(z^{(i)})\frac{-1}{\sigma} \tag{37}$$

$$\frac{\partial\left(q(z^{(i)})\right)}{\partial \mu} = \frac{\partial\left(q(z^{(i)})\right)}{\partial z^{(i)}}\frac{\partial z^{(i)}}{\partial \mu} = f(z^{(i)})\frac{1}{\sigma} \tag{38}$$

Making the substitutions results in

$$\begin{aligned}\frac{\partial^2 \mathrm{l}}{\partial \mu^2} = \sum_{\mathrm{i}=1}^{\mathrm{n}}\Bigg\{&\mathrm{N}^{(\mathrm{i})}\frac{-1}{\sigma}\left[\frac{1}{\mathrm{p}(\mathrm{z}^{(\mathrm{i})})}\mathrm{z}^{(\mathrm{i})}\mathrm{f}(\mathrm{z}^{(\mathrm{i})})\frac{1}{\sigma} + \mathrm{f}(\mathrm{z}^{(\mathrm{i})})\frac{1}{\left(\mathrm{p}(\mathrm{z}^{(\mathrm{i})})\right)^2}\mathrm{f}(\mathrm{z}^{(\mathrm{i})})\frac{1}{\sigma}\right] + \\ &\mathrm{M}^{(\mathrm{i})}\frac{-1}{\sigma}\left[\frac{-1}{\mathrm{q}(\mathrm{z}^{(\mathrm{i})})}\mathrm{z}^{(\mathrm{i})}\mathrm{f}(\mathrm{z}^{(\mathrm{i})})\frac{1}{\sigma} + \mathrm{f}(\mathrm{z}^{(\mathrm{i})})\frac{1}{\left(\mathrm{q}(\mathrm{z}^{(\mathrm{i})})\right)^2}\mathrm{f}(\mathrm{z}^{(\mathrm{i})})\frac{1}{\sigma}\right]\Bigg\}.\end{aligned} \tag{39}$$

And now, after some simplification the expression becomes

$$\begin{aligned}\frac{\partial^2 \mathrm{l}}{\partial \mu^2} = \sum_{\mathrm{i}=1}^{\mathrm{n}}\Bigg\{&\mathrm{N}^{(\mathrm{i})}\frac{1}{\sigma^2}\frac{1}{\mathrm{p}(\mathrm{z}^{(\mathrm{i})})}\mathrm{f}(\mathrm{z}^{(\mathrm{i})})\left[-\mathrm{z}^{(\mathrm{i})} - \frac{\mathrm{f}(\mathrm{z}^{(\mathrm{i})})}{\mathrm{p}(\mathrm{z}^{(\mathrm{i})})}\right] + \\ &\mathrm{M}^{(\mathrm{i})}\frac{1}{\sigma^2}\frac{1}{\mathrm{q}(\mathrm{z}^{(\mathrm{i})})}\mathrm{f}(\mathrm{z}^{(\mathrm{i})})\left[+\mathrm{z}^{(\mathrm{i})} - \frac{\mathrm{f}(\mathrm{z}^{(\mathrm{i})})}{\mathrm{q}(\mathrm{z}^{(\mathrm{i})})}\right]\Bigg\}.\end{aligned} \tag{40}$$

Applying the expectation operator and negating

$$\begin{aligned}-\mathrm{E}\left[\frac{\partial^2 \mathrm{l}}{\partial \mu^2}\right] = \sum_{\mathrm{i}=1}^{\mathrm{n}}\Bigg\{&\mathrm{E}[\mathrm{N}^{(\mathrm{i})}]\frac{1}{\sigma^2}\frac{1}{\mathrm{p}(\mathrm{z}^{(\mathrm{i})})}\mathrm{f}(\mathrm{z}^{(\mathrm{i})})\left[+\mathrm{z}^{(\mathrm{i})} + \frac{\mathrm{f}(\mathrm{z}^{(\mathrm{i})})}{\mathrm{p}(\mathrm{z}^{(\mathrm{i})})}\right] + \\ &\mathrm{E}[\mathrm{M}^{(\mathrm{i})}]\frac{1}{\sigma^2}\frac{1}{\mathrm{q}(\mathrm{z}^{(\mathrm{i})})}\mathrm{f}(\mathrm{z}^{(\mathrm{i})})\left[-\mathrm{z}^{(\mathrm{i})} + \frac{\mathrm{f}(\mathrm{z}^{(\mathrm{i})})}{\mathrm{q}(\mathrm{z}^{(\mathrm{i})})}\right]\Bigg\}.\end{aligned} \tag{41}$$

and noting that

$$E[N^{(i)}] = p(z^{(i)})(N^{(i)} + M^{(i)}) \tag{42}$$

$$E[M^{(i)}] = q(z^{(i)})(N^{(i)} + M^{(i)}) \tag{43}$$

after substitution the expression becomes

$$-E\left[\frac{\partial^2 l}{\partial \mu^2}\right] = \sum_{i=1}^{n}\left\{(N^{(i)} + M^{(i)})\frac{1}{\sigma^2}f(z^{(i)})\left[z^{(i)} + \frac{f(z^{(i)})}{p(z^{(i)})} - z^{(i)} + \frac{f(z^{(i)})}{q(z^{(i)})}\right]\right\}. \tag{44}$$

This expression is further simplified to its final form

$$a_{11} = -E\left[\frac{\partial^2 l}{\partial \mu^2}\right] = \sum_{i=1}^{n}\left\{(N^{(i)} + M^{(i)})\frac{1}{\sigma^2} f(z^{(i)})^2 \left[\frac{1}{p(z^{(i)})} + \frac{1}{q(z^{(i)})}\right]\right\}. \tag{45}$$

The a_{12} (same as a_{21}) and a_{22} terms are given as

$$a_{12} = a_{21} = -E\left[\frac{\partial^2 l}{\partial \sigma \mu}\right] = \sum_{i=1}^{n}\left\{(N^{(i)} + M^{(i)})\frac{z^{(i)}}{\sigma^2} f(z^{(i)})^2 \left[\frac{1}{p(z^{(i)})} + \frac{1}{q(z^{(i)})}\right]\right\}. \tag{46}$$

$$a_{22} = -E\left[\frac{\partial^2 l}{\partial \sigma^2}\right] = \sum_{i=1}^{n}\left\{(N^{(i)} + M^{(i)})\frac{\left(z^{(i)}\right)^2}{\sigma^2} f(z^{(i)})^2 \left[\frac{1}{p(z^{(i)})} + \frac{1}{q(z^{(i)})}\right]\right\}. \tag{47}$$

Next, these "a terms" (a_{11}, a_{12}, a_{22}) are plotted in figure 2.2.2 for a single specimen. This plot demonstrates that a_{11} and a_{22} cannot be maximized simultaneously. However, a good amount of information about μ and σ is achieved if the test is conducted at $\mu \pm \sigma$. Quoting directly from Neyer [2],

> A D-optimal result will be obtained when the determinant of the information matrix is maximized.
>
> ...
>
> Since the off-diagonal terms [a_{12} and a_{21} here] of the matrix are typically small compared to the diagonal terms, a D-optimal test will also approximately minimize the the product of the asymptotic variances of both μ and σ.

Thus, when choosing the next stimulus level to test in a sequential design of experiments, Neyer's algorithm elects the level which maximizes the determinate of the information matrix because this relates to minimizing the variances of $\hat{\mu}$ and $\hat{\sigma}$.

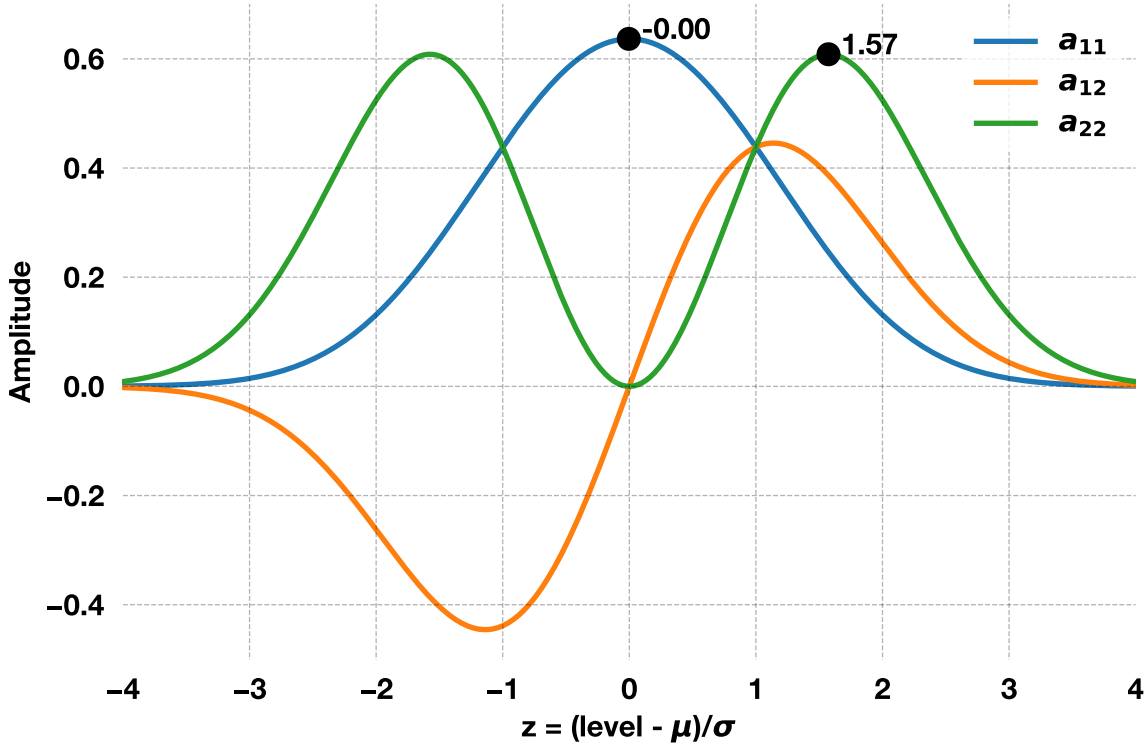


Figure 3: Values of the elements in the information matrix for a single specimen

Algorithm Flowchart

As noted, Neyer's sensitivity test chooses the stimulus level which maximizes the information matrix. However, in order to compute the information matrix, at least one point needs to be tested and estimates of μ and σ are required.

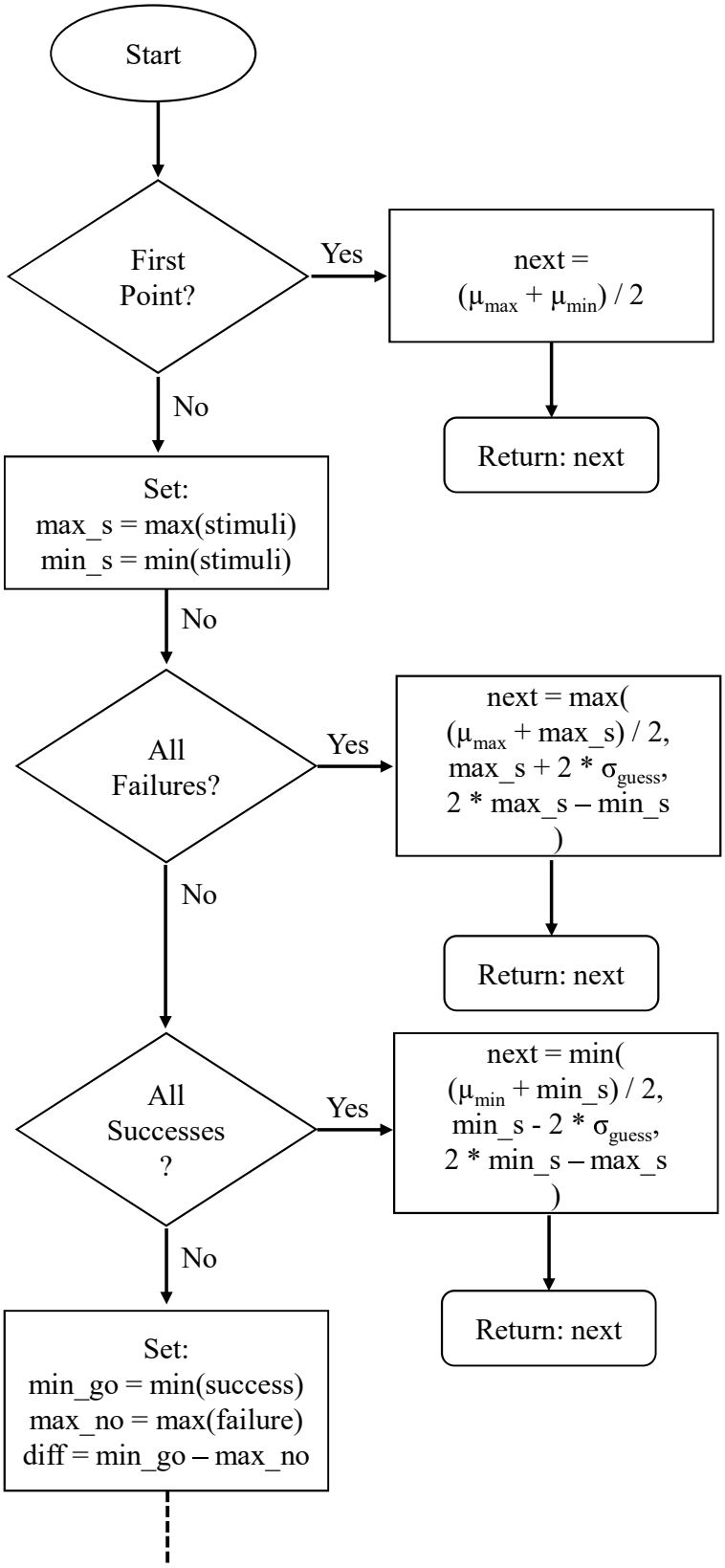
When beginning an experiment, it is possible that the experimenter has no prior knowledge about μ or σ; this scenario becomes more likely when testing a new material. Additionally, as mentioned, unique maximum likelihood estimators will only occur if there is overlap in the test outcomes. Neyer's algorithm is robust and overcomes these deficiencies and a brief summary of its is provided here.

To initialize the algorithm, the user needs only to provide a guess as to the lower and upper bounds of the expected μ and a guess for σ. The algorithm begins with an binary search between the given bounds. The bounds are updated so that lower bound becomes the maximum stimulus level that produced a negative response and the upper bound is the minimum stimulus level that provides a positive response. This procedure is repeated until difference in the bounds (upper bound minus lower bound) becomes less than the guess provided for σ. It should be noted that if the algorithm only detects a single type of response, the bounds are automatically adapted to increase the search length until mixed responses are observed.

Next, the algorithm uses the midpoint between highest stimulus level returned a negative response and the lowest stimulus level that returned a positive response as an estimate for μ. This μ along with the original guessed σ, σ_{guess}, and the data taken thus far (set of stimulus levels and their respective responses) are used to maximize the determinate of the information matrix. The stimulus level that maximizes the determinate of the information matrix is the next level tested. This procedure is repeated until overlap is achieved, and on each repetition σ_{guess} is updated by $\sigma_{\text{guess}} = 0.8 * \sigma_{\text{guess}}$. It should be noted that since σ_{guess} is decreasing, it is possible that the difference ([lowest stimulus that resulted in a positive response] - [highest stimulus that resulted in a negative response]) becomes greater than σ_{guess}. In this case, the algorithm resumes a binary search until this difference becomes less than the updated σ_{guess} or until overlap is achieved.

Once overlap is achieved, the guessed values for μ and σ are no longer used, but the maximum likelihood estimates are calculated. The algorithm performs a brief check to ensure that these estimates are not 'wild' and then uses these to determine the information matrix. Again, the stimulus level which maximizes the determinate of the information matrix is suggested as the next tested level. From this point, the algorithm continues on indefinitely until the experimenter as tested as sufficient number of specimens or until the set of specimens have been depleted.

As a visual aid, figure 4 gives a slightly adapted flowchart of Neyer's sequential design of experiments algorithm. In the original version, the maximum likelihood estimates of μ and σ are computed after the binary search has terminated regardless of whether overlap has been achieved. Without overlap, non-unique estimates are returned and $\hat{\sigma}$ should be zero. $\hat{\sigma} = 0$ is used as a decision flag: if true, then the current guesses of μ and σ are provided to the information matrix; and if false, the unique maximum likelihood estimates are passed to the information matrix (after 'wild' check, of course). However, I did not wish to rely on the MLE of σ to be zero as numerical precision or the choice of optimizer may preclude $\hat{\sigma}$ from being precisely zero. In my adapted algorithm, which is represented by the flowchart, the MLEs of μ and σ are only calculated if overlap is achieved. The occurrence of overlap is determined by simply looking at the difference between the highest negative level and lowest positive level; a quantity which is already calculated and stored. If this difference is negative, then overlap in the data is present. Without presenting proof, I believe this alteration will provide a more stable decision block between various coding languages, machine types, and choice of optimization routine.


Start
First Point?
Yes
next = (μmax + μmin) / 2
Return: next
No
Set: max_s = max(stimuli) min_s = min(stimuli)
No
All Failures?
Yes
next = max((μmax + max_s) / 2, max_s + 2 * σguess, 2 * max_s – min_s)
Return: next
No
All Successes ?
Yes
next = min((μmin + min_s) / 2, min_s - 2 * σguess, 2 * min_s – max_s)
Return: next
No
Set: min_go = min(success) max_no = max(failure) diff = min_go – max_no

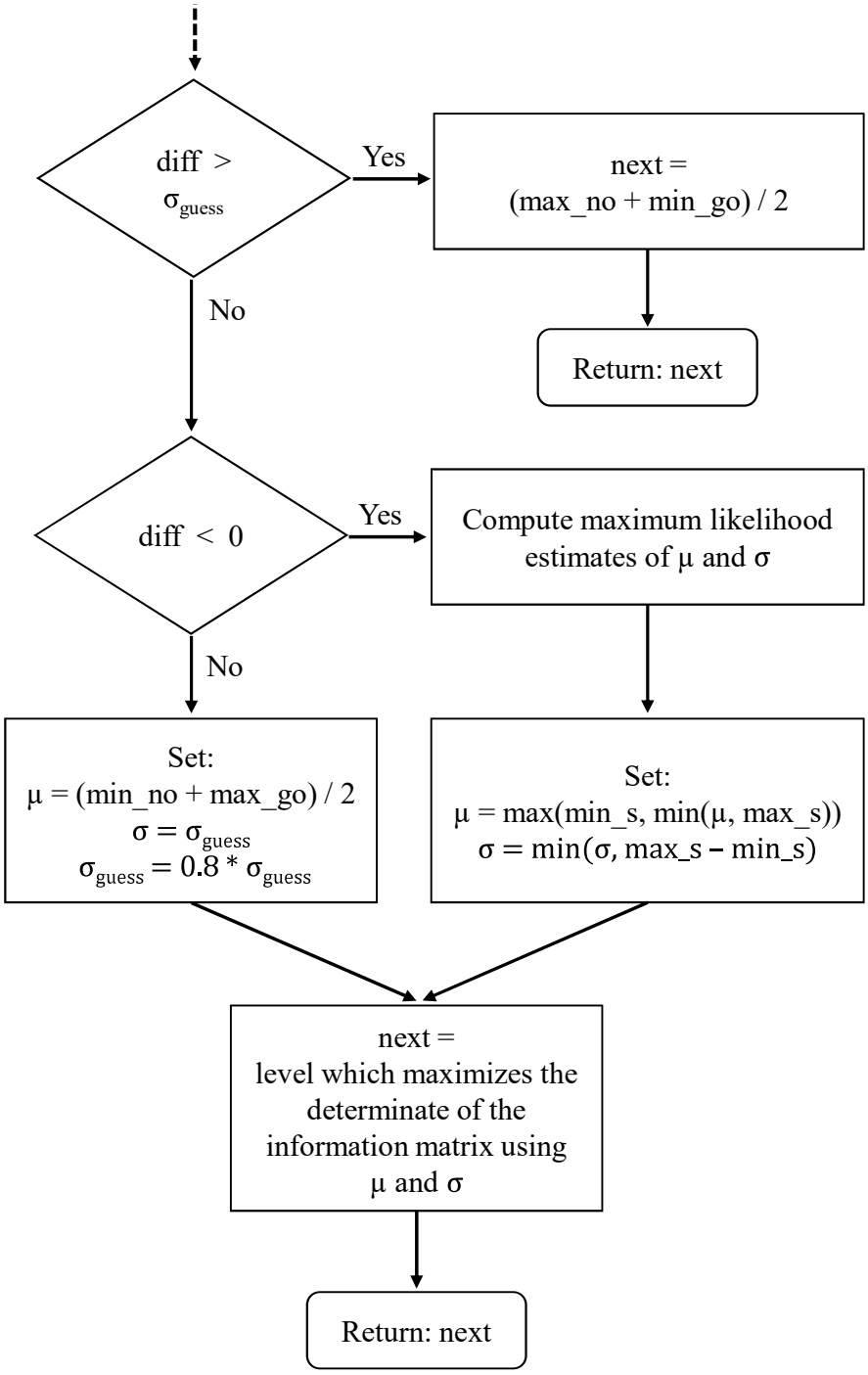


Figure 4: Flowchart of Neyer's sequential design algorithm

Equation Summary
Using the the normal distribution as an example, derivations for the first derivatives of the log-likelihood with respect to the parameters, the Hessian, and the expected information matrix were detailed. A summary of these relations and and those derived under the assumption that the latent distribution is logistic or log-logistic (not originally implemented by Neyer) are provided here.

Normal

todo

Logistic

todo

Log-logistic

todo

2.2.3 SenPy Implementation

The Neyer model is represented by a `Neyer` class which is located in the `neyer.py` module. Importing the module and creating the Neyer object can be done as shown:

```
from senpy.neyer import Neyer # import the Neyer ojbect from the neyer module
est = Neyer(latent='normal', inverted=False,
            method='L-BFGS-B', num_restarts=3,
            t1_min=None, t1_max=None, t2_guess=None,
            precision=8, resolution=None,
            lower_bound=None, upper_bound=None,
            hist=False, log_file=None) # instantiate the Neyer object and assign it to a
                                       variable named est (short for estimator)
```

Note that quite a few keyword arguments can be specified during initialization. However, none are required as sensible defaults are provided for all the them.

The Neyer documentation is as follows:

class `senpy.neyer.Neyer`**(*latent='normal', inverted=False, method='L-BFGS-B', num_restarts=3, t1_min=None, t1_max=None, t2_guess=None, precision=8, resolution=None, lower_bound=None, upper_bound=None, hist=False, log_file=None*)**
Bases: `object`

The Neyer model. Given an assumed form for the latent distribution, either 'normal', 'logistic', or 'log-logistic', the maximum likelihood estimates of the distribution parameters are computed. Neyer also provides a sequential design algorithm.

Parameters

- **latent** (*string,optional*) – DESCRIPTION. The form of the latent distribution. Either 'normal', 'logistic', or 'log-logistic'. The default is 'normal'.
- **inverted** (*boolean,optional*) – DESCRIPTION. If the probability of a 'go' increases as the stimulus level decreases, then the data is 'inverted'. The default is False.
- **method** (*string,optional*) – DESCRIPTION. Name of the optimization routine called when computing the maximum likelihood estimates. The default is 'L-BFGS-B'. Other options include 'SLSQP' and 'TNC'.
- **num_restarts** (*int,optional*) – DESCRIPTION. The number of random initializations to use when maximizing the likelihood function. Note, the available latent distributions only use two parameters. Consequently, the resulting likelihood function is typically convex. The default is 3.
- **t1_min** (*flaot,optional*) – DESCRIPTION. When using the sequential design algorithm and starting with no (or minimal) data, an intial guess on the lower bound of the first parameter, theta_1, is required. For the normal and logistic distributions theta_1 is mu. For the log-logistic distribution theta_1 is alpoha. If None is provided and the sequential algorithm is called, the program will prompt the user for the value. The default is None.
- **t1_max** (*float,optional*) – DESCRIPTION. The initial guess for the upper bound of theta_1. See t1_min for more details. The default is None.
- **t2_guess** (*float,optional*) – DESCRIPTION. The initial guess for theta_2. Required when using the sequential design algorithm. See t1_min for more details. For the normal and logisit distributions, theta_2 is sigma. For the log-logistic distribution, theta_2 is beta. The default is None.
- **precision** (*int,optional*) – DESCRIPTION. Number of decimal points to incude in the final output. The default is 8.
- **resolution** (*float,optional*) – DESCRIPTION. The smallest change in stimulus level available. For example, a drop-weight apparatus may only have adjustments at quarter inch intervals. Thus, the algorithm should not suggest testing at 12.105 inches, etc. The default is None.
- **lower_bound** (*float,optional*) – DESCRIPTION. The lowest stimulus level a user can phsically test. The default is None.
- **upper_bound** (*float,optional*) – DESCRIPTION. The highest stimulus level a user can phsically test. The default is None.
- **hist** (*boolean,optional*) – DESCRIPTION. If True the determinant of the information matrix is computed over a range of stimulus levels at each stage of the sequential design. Typically used for debugging only! The default is False.
- **log_file** (*str,optional*) – DESCRIPTION. File path for a log file. The log consists of the steps taken during the sequential design algorithm. The default is None.

`fit`**(*X, Y*)**
Compute the maximum likelihood estimates of the distribution parameters.

Parameters

- **X** (*2D array*) – The tested stimulus levels. Must be of shape (n_pts, 1)
- **Y** (*array*) – The observed response at each stimulus level. 1 for 'go' and 0 for 'no-go'.

Returns

Return type self

`get_estimators`**()**
Provides access to the stored estimate of theta. For example, [mu, sigma] or [alpha, beta].

Returns Current parameter estimates. Shape is (2,)

Return type array

`loop`***(iterations=1000000)***
This method suggests new test levels and accepts user input to calculate maximum likelihood estimates. That is, this method constitutes a loop. Loop will continue indefinitely until 'end' is received as user input during the either the test level or result input queries. Alternatively, if a set number of specimens is to be used then the number of loops can be specified with the 'iterations' keyword argument.

Parameters

- **iterations** (*int,optional*) – End the loop automatically after n iterations. The default is 1000000.

Returns

Return type None.

`next_pt`**()**
The sequential design algorithm. When this method is called, the next suggested stimulus level for testing is printed to the console.

Returns

Return type self

`plot_confidence_region`***(limits, n, CI_levels=10, save_dst=None, show=True)***
A high-level function to plot the confidence region of the parameters.

Parameters

- **limits** (*list*) – The plot limits provided as [lower xlim, upper xlim, lower ylim, upper ylim].
- **n** (*int or list of length 2*) – The number locations to sample in the x (theta_1) and y (theta_2) directions.
- **CI_levels** (*int or list,optional*) – If an integer, a filled contour plot will be produced with that many levels. If it is a list, the list values specify the confidence levels at which to draw contour lines. The default is 10.
- **save_dst** (*str,optional*) – The file path (including file type) where the plot should be saved. The default is None
- **show** (*boolean,optional*) – If True, simply calls matplotlib.plt.show(). May be required for some IDEs. The default is True.

Returns

Return type None.

plot_probability(***include_data=True*, *xlabel=None*, *ylabel=None*, *alpha=1.0*, *save_dst=None*, *show=True*, ***kwargs***)
A high-level method to call self.predict_probability and plot the result.

Parameters

- **include_data** (*boolean,optional*) – Whether or not to plot the data (stimuli and responses). The default is True.
- **xlabel** (*str,optional*) – If provided, the text for the plot xlabel. The default is None.
- **ylabel** (*str,optional*) – If provided, the text for the plot ylabel. The default is None.
- **alpha** (*float,optional*) – opacity of the observed data points. Must be between 0 and 1. Only used if include_data is True. Useful to visualize many overlapping data points. The default is 1.0.
- **save_dst** (*str,optional*) – The file path (including file type) where the plot should be saved. The default is None.
- **show** (*boolean,optional*) – If True, simply calls matplotlib.plt.show(). May be required for some IDEs. The default is True.
- ****kwargs** – All keyworkd arguments provided to predict_probability can also be provided here.

Returns

Return type None.

post_test_outcome(***res*, *pt***)
Append a stimulus level and result to the existing data.

Parameters

- **res** (*int or boolean*) – The observed result at the tested stimulus level. Either 0, 1 or False, True.
- **pt** (*float*) – The stimulus level at which the test was performed.

Returns

Return type None.

predict_probability(***pts=None, confidence=None, CI_level=[0.5, 0.8, 0.9, 0.95], num_samples=1000, max_iter=5***)
Returns the probability of a 'go' at pts. p(y=0|pt)

Parameters

- **pts** (*array,optional*) – The stimulus levels at which to compute probability predictions. The default is None. If None, range = max(X) - min(X) and pts = np.linspace(min(X)-0.5*range, max(X)+0.5*range, 100)
- **confidence** (*str,optional*) – The name of the method used to supply confidence intervals. Options are 'delta', 'perturbation' (same as delta), 'likelihood-ratio', 'parametric-bootstrap', and 'nonparametric-bootstrap'. The default is None.
- **CI_level** (*list,optional*) – The confidence levels. Ignored if confidence is None. The default is [.5, .8, .9, .95].
- **num_samples** (*int,optional*) – The number of bootstrapped samples generated. Only used if confidence = 'parametric-bootstrap' or 'nonparametric=bootstrap'. The default is 1000.

- **max_iter** (*int,optional*) – The maximum number of attempts to map the likelihood ratio. Only used if confidence = 'likelihood-ratio'. The default is 5.

Returns Consists of the stimulus points, the predicted probability, and arrays of the lower bounds and upper bounds of the confidence levels if confidence was requested. (pts (n_pts, 1), predicted probability (n_pts, 1)) or (pts (n_pts, 1), predicted probability (n_pts, 1), lower CI bounds, upper CI bounds) where the shape of lower and upper CI bounds is (n_pts, n_levels)

Return type tuple

`print_estimators`**(*cost=False*)**

Prints the current parameter estimates to the console.

Parameters

- **cost** (*boolean,optional*) – If true, the value of the negative log-likelihood, or cost, at the current parameter estimates is also printed to the console. The default is False.

Returns

Return type None.

2.3 Dror-Steinberg (a Bayesian approach)

IN DEVELOPMENT

2.3.1 Background

2.3.2 Model Overview

2.3.3 SenPy Implementation

2.4 Isotonic regression

IN DEVELOPMENT

2.4.1 Background

2.4.2 Model Overview

2.4.3 SenPy Implementation

2.5 Gaussian process classifier

IN DEVELOPMENT

2.5.1 Background

2.5.2 Model Overview

2.5.3 SenPy Implementation

3 Code

3.1 Repository Contents

The root repository is named `senpy`. This contains:

1) SenPy_Manual.pdf (this document)
2) LICENSE – a standard MIT license agreement
3) README.md – a document written in markdown which provides a basic overview of the package
4) requirements.txt – list of packages required for SenPy to run
5) **examples** – a directory containing scripts which demonstrate basic usage of SenPy
6) **senpy** – the actual SenPy package: a directory of python modules
7) **docs** – directory of html code to display the documentation; either click on the `index.html` file or view the documents hosted here

3.2 Requirements

All required packages and their specific version numbers are listed in `requirements.txt`. It is suggested that you create a virtual environment and set it up with the required packages using

```
pip install -r requirements.txt
```

or

```
conda install -r requirements.txt
```

if using conda. However, the dependencies used are *fairly* commonplace and matching the exact version number should not matter in most cases – as such, it is quite possible that SenPy will run using your current python installation (or with minimal tweaking).

Current requirements:

certifi==2019.11.28

cloudpickle==1.3.0

cycler==0.10.0

cytoolz==0.10.1

dask==2.12.0

decorator==4.4.2

imageio==2.8.0

kiwisolver==1.1.0

matplotlib==3.1.3

mkl-fft==1.0.15

mkl-random==1.1.0

mkl-service==2.3.0

networkx==2.4

numpy==1.18.1

olefile==0.46

Pillow==7.0.0

pyparsing==2.4.6

python-dateutil==2.8.1

PyWavelets==1.1.1

scikit-image==0.16.2

scipy==1.4.1

six==1.14.0

toolz==0.10.0

tornado==6.0.4

wincertstore==0.2

3.3 Using the Code

In the `senpy` repository the subdirectory `senpy` is the python package. Download and add this directory to your current working directory, python site-packages, or add its location to your python path.

Ensure that all dependencies are installed. See section 3.2 for more details.

You should now be able to import the senpy package. Test this by running the following commands in a python console or in a script:

```
import senpy
print(senpy.__version__)
```

Assuming you are using the code version that corresponds to this manual, you should see `'1.2'` in the output. If so, you are now ready to use the package.

Each sensitivity method/algorithm resides in its own module. For example, the Neyer method is found within the neyer.py module. Each method's module defines a class named after the method. And each method's functionality is completely encapsulated by its class. So, in order to use functions associated with the Neyer algorithm you instantiate a Neyer object like so:

```
from senpy.neyer import Neyer # import the Neyer ojbect from the neyer module
est = Neyer() # instantiate the Neyer object and assign it to a variable named est (short for
    estimator)
```

Now, using `est` you can estimate the latent distribution parameters by passing the data (collection of tested stimulus levels and the observed responses, 'go' or 'no-go') to `est.fit(stimuli, responses)`. Additional attributes and functionality can be accessed through the object – see each method's documentation for more details (section 2).

Although the Neyer method was used as an example here, the procedure is similar for all of the algorithms in SenPy. Specific documentation for each method can be found under the **SenPy Implementation** headings in section 2. Documentation is also provided online here.

3.4 Caveats

1) None

3.5 Contact

Bug reports and general questions can be sent to alex.casey.13 at gmail.com.

4 Minimal Working Examples

4.1 Neyer

The `examples` folder contains a file named `mle_neyer.py`. This file demonstrates how the Neyer method can be used to calculate the maximum likelihood estimates for the distribution parameters from data. `mle_neyer.py` is reproduced in its entirety here:

```
import numpy as np
from senpy.neyer import Neyer

# this is the example data provided in the Neyer paper
# X_data must be of shape [n_pts, 1]
x_data = np.array([1, 1.2, 1.4, 1.8, 2.6, 4.2, 3.4, 3.8, 4, 4.1, 4.28,
                   4.52, 5.55, 5.24, 6.37, 6.08, 7.38, 7.09, 6.89, 6.74]).reshape((-1,1))
y_data = np.array([0,0,0,0,0,1,0,0,0,0,0,0,1,0,1,0,1,1,1,1])

# instantiate a Neyer object. In this case, we wish to assume a log-logistic
# latent distribution
est = Neyer(latent='log-logistic')

# call fit method on the Neyer estimator
est.fit(x_data, y_data)

# plot the results
# show=True simply call matplotlib.pyplot.show() which may be necessary
# depending on your IDE
est.plot_probability(confidence='likelihood-ratio',
                     xlabel='Drop Height (m)', ylabel='Predicted Probability',
                     show=True)
est.plot_confidence_region([4, 7, 1, 20], [100], show=True)

# print the parameter estimates to console
est.print_estimators()
```

Output:

```
alpha: 5.317603889844265
beta: 8.27725905945797
```

This code will also produce figures 5 and 6.

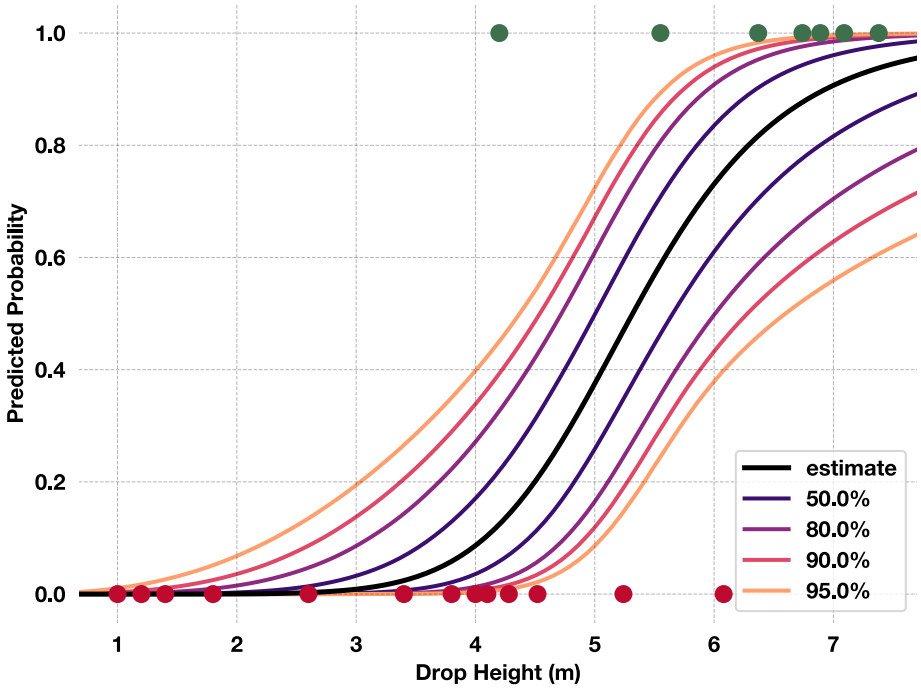


Figure 5: Test results and predictive probability with confidence intervals from an explosives drop-weight impact test.

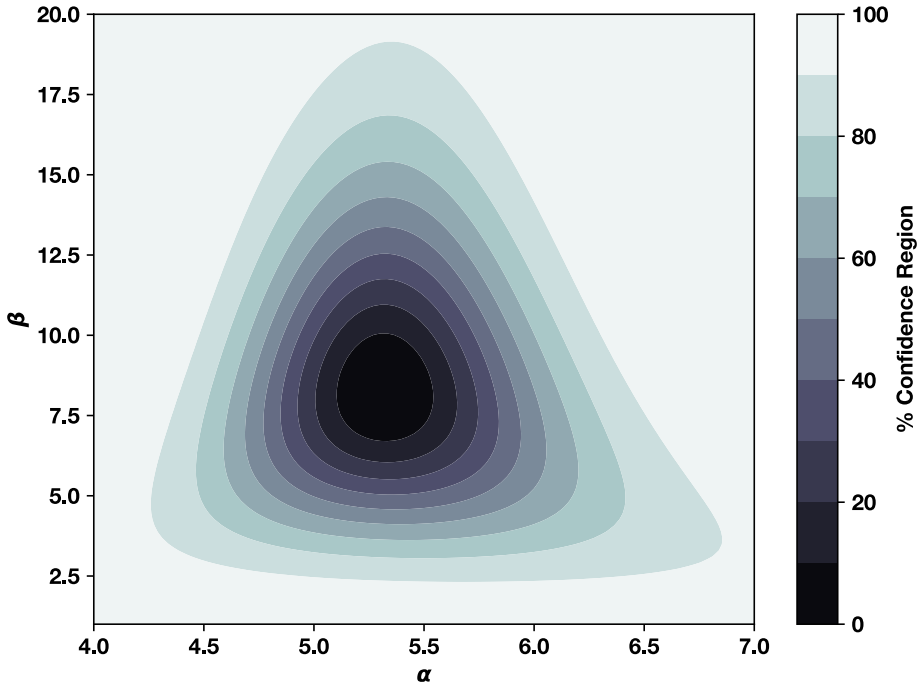


Figure 6: Confidence regions on the log-logistic distribution parameters α and β.

5 References

[1] B. T. Neyer, “More efficient sensitivity testing,” tech. rep., EG and G Mound Applied Technologies, Miamisburg, OH (USA), 1989.

[2] B. T. Never, “A d-optimality-based sensitivity test,” *Technometrics*, vol. 36, no. 1, pp. 61–70, 1994.

[3] J. Cornfield and N. Mantel, “Some new aspects of the application of maximum likelihood to the calculation of the dosage response curve,” *Journal of the American Statistical Association*, vol. 45, no. 250, pp. 181–210, 1950.

[4] A. Golub and F. E. Grubbs, “Analysis of sensitivity experiments when the levels of stimulus cannot be controlled,” *Journal of the American Statistical Association*, vol. 51, no. 274, pp. 257–265, 1956.

[5] W. J. Dixon and A. M. Mood, “A method for obtaining and analyzing sensitivity data,” *Journal of the American Statistical Association*, vol. 43, no. 241, pp. 109–126, 1948.

[6] D. Finney, “Probit analysis, cambridge university press,” *Cambridge, UK*, 1947.

[7] C. I. Bliss, “The calculation of the dosage-mortality curve,” *Annals of Applied Biology*, vol. 22, no. 1, pp. 134–167, 1935.

[8] C. I. Bliss, “The calculation of the dosage-mortality curve,” *Annals of Applied Biology*, vol. 22, no. 1, p. 149, 1935. Appendix by R. A. Fisher.

[9] M. J. Silvapulle, “On the existence of maximum likelihood estimators for the binomial response models,” *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 310–313, 1981.