FAST: Functional Adaptive Sequential Testing.

Ed Vul, Don MacLeod
Contact: evul@mit.edu

This is a Matlab toolbox for efficient estimation of thresholds in psychophysics experiments.

FAST is 'functional', that is, you can estimate threshold functions (such as contrast sensitivity as a function of frequency). Other methods focus on estimating one threshold at a time. By using the functional relationship between different points along the function, we make what would be a number of independent, slow staircases, converge faster by informing each by the others. This also allows you to sample every point along the function, rather than a few discrete points, thus giving you a better picture of the function as a whole, and where your presumed functional form may be wrong.

FAST allows you to do the following:

1. Estimate thresholds and width/slope.
2. Estimate thresholds as a function of some other stimulus parameter
3. Choose stimuli for different estimation goals.
4. Shrink, move, and otherwise adaptively alter the search parameter space.
5. Implement elaborate stopping rules.
6. Set priors on all of your parameters

Because FAST can be used to achieve several different ends, its interface may be complicated in parts. We've tried to make the additional complexities of FAST unnoticeable to those using its basic functions, but at points, some things about basic usage might be confusing because they also have to serve more complicated uses. All that said, let's dive in:

Overview:

To use FAST in an experiment, you have to

1. Create a FAST structure,
2. Choose a stimulus
3. Update the FAST structure with the response and stimulus values for the particular trial.
4. Repeat 2-3 until stopping.

Each of these steps can be very simple, or very complicated, depending on how much you want to get out of FAST.

As a basic overview:

FAST is designed to estimate thresholds.

Particularly, it is most useful for estimating thresholds at different points of some function.

For instance: a contrast sensitivity function:

Let Y* be the threshold detection contrast

Let X be the spatial frequency

Let P be the probability of responding 'Yes' on the detection task.

Under the FAST framework:

P=F(Y*,Y,thetaPsy)

Where F is some psychometric function, Y is the presented contrast, and thetaPsy is the parameters of the psychometric function.

Y*=G(X, thetaFunc)

Where G is some function with parameters thetaFunc that is adopted as a working approximate description of the relation of X (say spatial frequency) to the threshold value (say contrast).

Thus, we assume that the psychometric function is invariant to translations along the X dimension. Using this formulation, we can more efficiently sample points in the X,Y plane to determine the parameters of G and F.

Function List

fastInit() Makes a FAST structure.

fastPredict() Predicts the stimulus value that will generate a response with a particular probability.

fastChooseYp() Like fastPredict, but integrate over all parameters.

fastChooseYent() Choose (local) expected entropy minimizing Y value for a given X.

fastChooseYvar() Choose (local) expected probability variance minimizing Y value for a given X.

fastChooseXY() Choose (global) expected entropy minimizing X and Y value.

fastUpdate() Updates the FAST structure with new data

fastResample() Shift the parameter grid.

fastSetPrior() Set or change the prior over parameters.

fastPlot() Plots the current results.

fastEstimate() Estimates the parameter values with higher fidelity than is possible online.

Very basic usage
(illustrated with the usual single value threshold estimation)

What you need to know:

1. Some basics about the task, and the shape of the psychometric function:
   A. What is the task (yes/no, matching) or nAFC?
   B. What type of psychometric function do you want to use to represent the variation of response probability with stimulus value?
   C. What is a reasonable estimate of the width/slope? The meaning of this parameter varies with the psychometric function chosen and with the way that the adaptively varied stimulus value is specified. In detection tasks, it is often convenient to adopt a psychometric function that expresses the probability of detection as a function of a logarithmic measure of stimulus strength. This function can be shifted along the logarithmic abscissa to estimate the threshold, with chance performance occurs at an abscissa value of –Inf. FAST's default psychometric function, the logistic function, $p = 1/(1+\exp(-(s-s^*)/w)$ is of this type and presupposes that s is a logarithmic measure of stimulus strength. The shape of the psychometric function is mainly determined by its width or slope. The width is fixed by a second parameter, *w*, distinct from the 50% threshold *s**, but an appropriate estimate of *w* improves the fit and the threshold estimate. The interquartile range is roughly 2w; $s = s^* + w$ gives $p = .73$; $s = s^* - w$ gives $p = .27$. Where *s* is the decimal logarithm, a plausible initial guess is that *w* will be around 0.1.
2. Some estimate of the function and function parameters you want to be estimating (in the basic case, this would just be a threshold). The more you know and specify up front, the faster the adaptive procedure will converge.

1. Setting up the FAST structure

```
function fast = fastInit(funccurve, nchoice, funcpsych, parameters)
```

funcpsych is the psychometric function, in this case, 'psyWeibull
nchoice is the task: 0 or 1 means matching/detection, 2+ is nAFC
funccurve is the function we are estimating, in this case a singular threshold: 'funcVal'
parameters is a cell array of parameters, specified most simply as a range for each.
The last parameter is the psychometric function slope, the first N-1 are the funccurve parameters

```
myFast = fastInit('funcVal', 0, 'psyLogistic', {[1 100] [1.5 4.5])
```

So we've set up a single threshold estimation where we estimate the threshold to be between 1 and 100 units, with a detection task (0 choice) using a Weibull function with width/slope (beta exponent for the Weibull) estimated to be between 1.5 and 4.5 units.

2. Choose stimulus value.
Y=fastChooseYent(fast, x, yrange)
This predicts the Y value (within y range) that will minimize expected posterior entropy given that the x value must be x. This is an efficient, simple, and speedy way to choose stimuli, but other methods are available.

```
Y = fastChooseYent(myFast, 0, [1 100])
```

This will give us a Y value that should be most informative of all of the stimulus parameters.

.

3. Update FAST
fast = fastUpdate(fast, [x y r])

This updates the current likelihoods and best parameter estimates for the FAST structure (fast) and the newly received data [x y r]: a trial at x value X, y value Y, and a response of r.

So, if we had just presented the trial described in step 2, and the subject said that they did detect the stimulus (r=1), we would update the structure with the following code:

```
myFast = fastUpdate(myFast, [0 Y 1])
```

And thus we can continue reiterate steps 2 and 3 until we are satisfied.

That's it.
That is all you need to do to run an experiment using FAST. However, plenty other options are available. For details of each, please see help of the appropriate function. What follows next is a brief overview of the possibilities.

Full functioned usage

1. Setting up the FAST structure

When defining the fast structure, several more possibilities are available, mostly by expanding the parameter cell to specify more details.

Expanding on parameters:

Most usefully, you can specify (rather than have FAST infer) whether a parameter is linear or log spaced (NOTE: psychometric slope parameter must be logarithmically spaced!).

Second, you can specify the range in terms of a prior (log or linear normal) by providing a mean and a standard deviation.

Third, you can specify the sampling density for each of the parameters (although there is probably no good reason to do this manually).

Providing x,y limits:

To use fastChooseXY (global entropy minimization), we must define a conditional probability look-up table to make the necessary computations speedy enough to be done on each trial. To do so, you need to provide a range of x and y values we should consider.

2. Stimulus selection

Stimulus selection can be done in a number of different ways, each is done by a different function:

fastChooseYent: local (find y for a given x) entropy minimization.

fastChooseYvar: local minimization of posterior probability variance.

fastChooseXY: global (find both x and y) entropy minimization

fastChooseYp: for a given x and p, find a y that will produce a positive response with probability p (integrating over parameter space).

fastPredict: for a given x and p, find a y that will produce a positive response with probability p, using some particular set of parameters.

3. Update

fastUpdate can also output a second term: resamp, if this is 1, resampling is suggested:

fastResample: resamples parameter space by some combination of shifting or shrinking the current parameter lattice. This procedure can circumvent the problem of grid-search in high dimensional spaces (the inherent tradeoff between sampling range and sampling density).

Check help of fastResample for details of optional parameters, here I will provide just the intuition of what the function does: it estimates the most likely parameter values (via interpolation) as well as the standard deviation of the parameter likelihoods. Then it shifts the grid to be centered on the most likely parameter and span several (default 3) standard deviations on either side of the most likely parameter. Because of computational imperfections when estimating likelihoods from grids (or even in general), the most likely parameter and standard deviations may be misleading before sufficient

data are gathered, as such, it is ill-advised to resample before 10*N (where N is the number of parameters) data points have been collected. As further safe-guards against numerical errors in this estimation process, the center of the parameter range will not be moved further than the one half of the range outside the range, and the current breadth of sampling will only decrease (if necessary), but never increase. This means that it is advantageous to start off with broader, rather than narrower estimates of your parameters during initialization.

Since resampling the entire parameter space means re-evaluating each of the previous data-points again, this process may take several seconds (depending on how many parameters and data points there are). As such, it is probably best to only resample during periods when no time would be wasted by a pause of several seconds – e.g., when subjects are reading instructions.

4. Stopping rules
One can implement fancier stopping rules given the data saved in the FAST structure. In fast.params.est you will find cell arrays of the current estimates of the parameters, as well as confidence intervals and covariance matrices for the log-likelihood function. The confidence intervals may be used to implement stopping rules based on uncertainty.

5. Offline estimation, plotting, and fitting
fastPlot() plots the current data and best function fits, as well as any systematic deviations that may occur in the responses. Check help fastPlot() for details.

fastEstimate() allows for slower but more informative off-line estimation of function parameters. While we do not advise the use of fastEstimate() for thorough analysis (since we think it is rather important that you know what your analysis is doing) fastEstimate() can provide confidence intervals on all parameters, as well as the covariance structure and plots of marginals and likelihood surfaces. We have found these to be moderately useful. Please check help of fastEstimate for details.

6. Customizing
We have included functions for which we at one point had some use. You may have other functions you are interested in. It is relatively easy to write function function (e.g., funcLine), it is trickier to write psychometric functions, as they are overloaded and must operate in several different modes. Look at the code of (e.g., psyLogistic) for details of psychometric function overloading. Look at the code of (e.g., funcLine) for details of how parameters are specified and values are output from functions.

If you have any questions, please email me:
evul@mit.edu