

Program Manual for State-Space Modeling Toolbox (COMPASS)

Oct 2018

In this manual, we start with a sample problem which describes the pipeline of processing using the toolbox functions. We then provide a detailed explanation for each function used in this example.

Example Problems

The data file in this example is a [mat](#) file called [LEARNING_1.mat](#) which can be found in the [GitHub](#) repository. This file includes the behavioral outcome per trial, including both reaction time and binary (correct/incorrect) decisions. The data come from an associative learning task, where through the task, a rhesus macaque learns the correct response on each trial to receive reward. For this task, we are interested in analyzing how the animal learns over time. The source code for this example is called [compass_example_step_by_step.m](#) and can be found in the [GitHub](#) folder.

We can import the data by calling the following lines:

```
% Load behavioral data and prepare it for the toolbox
% Data: Yb (decision), Yn (reaction time)
load('LEARNING_1.mat');
% Yn - logarithm of reaction time in seconds
Yn = log(Yn/1000);
% Data length
N = length(Yn);
% Input - 1 xi
In = zeros(N,2); In(:,1)= 1; In(:,2)= 1;
% Input, Ib is equal to In
Ib = In;
% Uk, which is all zero
Uk = zeros(N,1);
% valid, which is valid for the observed data point
ind = find(isnan(Yn));
Valid = ones(N,1);
Valid(ind)=0;
```

Figure 1 shows the reaction time and correct/incorrect response per trial:

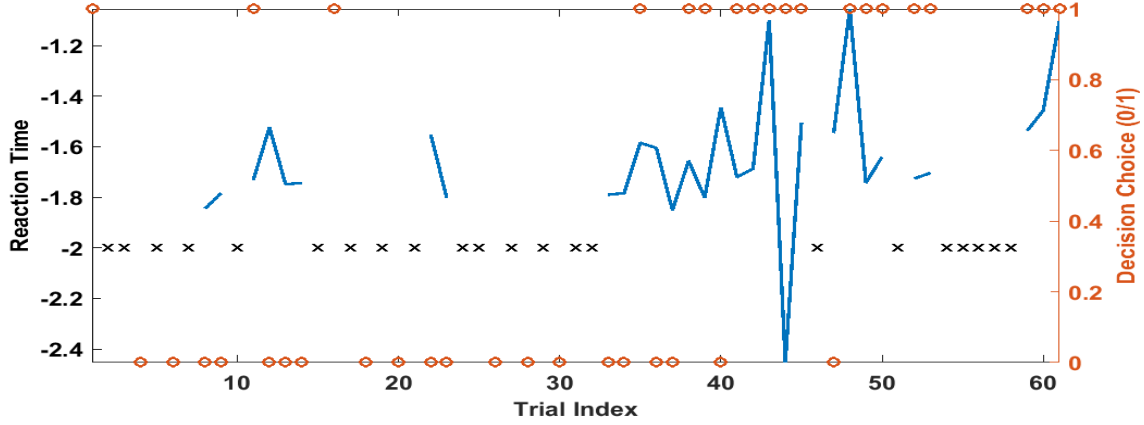


Figure 1 Reaction time and choice per trial. Trials where no response was made are marked by 'x'.

There is a clear trend over time towards more correct responses, showing that the animal learns the task. It also shows as the animal learns the task, it requires a longer time to pick the correct choice. Our hypothesis is that the underlying mental processes – the learning state – is projected on both the reaction time and the decision. Our goal is to infer the underlying learning state through the behavioral outcome. We also assume that in the missed trials, the animal is not focusing on the task, and as result, these points don't carry information about the learning state. To model this task, we assume that the learning state evolution over trials can be described by a random walk model.

$$x_{k+1} = x_k + w \quad v \sim N(0, \sigma_w^2) \quad (1)$$

where, x_k is the learning state over trial k . The noise term allows the state to change from one trial to the next. The observation per each trial is defined by $z_k = \{(y_{c,k}, y_{b,k}), 'Missed trial'\}$. The observation model for observed trials is defined by

$$p(y_{b,k} = 1) = \exp(x_k + c_0) / (1 + \exp(x_k + c_0)) \quad (2.a)$$

$$\log y_{c,k} = b_1 x_k + b_0 + v \quad v \sim N(0, \sigma_v^2) \quad (2.b)$$

where, $y_{b,k}$ is decision choice per trial. $y_{b,k}$ is either 1 which means a correct choice or 0 which means an incorrect choice. $y_{c,k}$ is the reaction time which is defined in seconds. We assume that probability of a correct choice follows a Bernoulli process and can be defined by a sigmoid function. Given the model definition, as x_k increases, the probability of picking the correct choice increases. c_0 defines the probability of picking the correct choice in the absence of the learning state; this might be biased and it is not necessarily always 50/50. We also assume that the reaction time can be defined by a log-normal distribution, and thus that the logarithm of reaction time is defined by a linear function of the learning state. b_0 defines the reaction time baseline and b_1 defines the slope of reaction time as a function of the learning state. Given the reaction time trend over trials (increasing with learning), we expect b_1 to be a positive number and b_0 to be negative. We also model the reaction time variability - or the other unknown mechanism in reaction time - using an additive white noise v . The toolbox helps us to estimate the model parameters - $c_0, b_1, b_0, \sigma_v, \sigma_w$ - along with the learning state x_k . Note that there is no scaling term (which might be called c_1 if it were present) for x_k in the Bernoulli process. Adding this term would lead to a model that is too complex and not identifiable. x_k appears in both the reaction time and decision choice formulas. If we have $c_1 x_k$ and $b_1 x_k$ terms rather than x_k and $b_1 x_k$, then we have infinitely many choices for x_k, c_1 and b_1 - for example, $\alpha^{-1} c_1$ αx_k and $\alpha^{-1} b_1$ αx_k are valid for any nonzero values of α .

Now that we have the model definition, we can implement the model using the COMPASS toolbox. The first step is defining the structure of the model. The following line in the code builds the model

```
% Build behavioral model and learning procedure
% create model
Param = compass_create_state_space(1,1,2,2,1,1,1,1,0);
```

Look at the [compass_create_state_space](#) function description to see each of its input argument definitions and how they are set in this example. We have one state, so `nx` is 1. We have no inputs (trial types or external perturbations), so we set `nUk` to 1, and we set all of `Uk`'s elements zero in the code above. This corresponds to the random walk definition of the learning state. We also assume `ln` and `lb` dimensions (`nln` and `nlb`) are 2 and the elements of `lb` and `ln` are all set to 1. This implies the specific form of the observation process defined in equation (2). The COMPASS code assumes models of the form:

$$\log y_{c,k} = b_1 \ln(k, 1) x_k + \ln(k, 2) b_0 + v \quad (3)$$

and

$$p(y_{b,k} = 1) = \exp(lb(k, 1) x_k + lb(k, 2) c_0) / (1 + \exp(lb(k, 1) x_k + lb(k, 2) c_0)) \quad (4)$$

that is, why `nln` and `nlb` are set to 2 in this example. To have our specific model as defined in equation set (2), we set all elements of `lb` and `ln` to 1.

We also address the identifiability issue by setting the last argument of the function to 0, which implies the specific form of discrete observation process- $c_1 = 1$. Note that, a more general form of binary observation is defined by

$$p(y_{b,k} = 1) = \exp(c_1 x_k + c_0) / (1 + \exp(c_1 x_k + c_0)) \quad (5)$$

where, we set c_1 to 1. Note that it is also possible to fix b_1 at 1 and let c_1 be adjusted by the model. If we use this definition, the interpretation of the learning state might change given the model parameters. Here, we have set `dLinkUpdate` to 0 and `cLinkUpdate` to 1 – see [compass_create_state_space](#) for the definition of these parameters. We can switch these values if we want to set b_1 at 1 and have c_1 learned from data instead.

Now that we have defined the model using [compass_create_state_space](#), we can define the model learning rule. The following lines in the code define the learning setup.

```
% set learning parameters
Iter = 250;
Param = compass_set_learning_param(Param, Iter, 0, 1, 0, 1, 1, 1, 1, 0);
```

We can follow the [compass_set_learning_param](#) function description to see each of its input argument definition and how they are set in this example. In this example, we set the number of EM iterations to 250. These input arguments also cause the state-transition parameters to be fixed and not learned from data. Note that, the general structure of the state transition model is:

$$x_{k+1} = a x_k + b * u_k + w \quad w \sim N(0, \sigma_w^2) \quad (6)$$

where here, a is set to 1 and is not updated. That is, x_k does not decay or increase on its own. If we wished to model some form of forgetting or imperfect working memory, we might set a to some number slightly less than 1. We already set u_k to zero. The arguments above cause the noise term to be updated based on data, and we assume x_0 is fixed and not updated here. Also, the input arguments suggest that other parameters of the model will be adjusted during the training. This setting is in accordance with our model specification, and it can be changed if the structure of the model or our assumption about the model parameters will change. The last term is set to 0, and would only be non-zero in models using gamma distributions.

Finally, the data we are using, like many other datasets, contain missing observations. These are not necessarily missing at random – they might indicate times when the animal did not decide before a timer expired. We need to identify these "censored" points for the algorithm and tell it what to do with them. We call the following function to define the censoring threshold and the methodology to address censored data points in the data.

```
% define censored point threshold
Param = compass_set_censor_threshold_proc_mode(Param,log(2),1,1);
```

That is, we define trials above 2 seconds to be censored, and would attempt to impute them by a single imputation draw. However, above (and below), we set elements of `obs_valid` as 0, implying they are missing at random. Those data will not be imputed regardless of this function's settings. If we instead set elements of `obvs_valid` to 2, the imputation specified in this function will take effect.

```
% valid, which is valid for the observed data point
ind = find(isnan(Yn));
Valid = ones(N,1);
Valid(ind)=0;
```

Note that we have defined the data attributes when the data were loaded. We define a `Valid` vector in the data loading process, which carries information about the data. The elements of `Valid` is 0, 1, or 2. An observed data point comes with `Valid` value of 1. Value 0 means a MAR data point and 2 means a censored data point. Note that the Value elements for dropped data points are set to 0. If we set `Valid(ind)=2`, then we assume the data on these trials are censored with a time threshold of 2 seconds. For other arguments check `compass_set_censor_threshold_proc_mode` definition.

Note that the model definition and learning information are all kept in the `Param` structure. The definition of `Param` elements can be found in the following pages.

Till this point, we have defined the model structure in COMPASS and we have set the learning rule. We have prepared data for the toolbox and set its attributes. The next step is to call the COMPASS main function, `compass_em`. The following line of code calls the function

```
%% Run learning with a mixture of normal & binary
[XSmt, SSmt, Param, XPos, SPos, ML, YP, YB]=
    compass_em([1 1], Uk, In, Ib, Yn, Yb, Param, Valid);
```

The function input includes the `Param`, type of observation process, and the data observations. Note that in this example, we utilize both continuous (reaction time) and discrete observations in estimating the

learning state. This corresponds to setting the `DISTR` argument to `[1 1]`. We might change this argument to other values if we change our assumption about the observation process. For example, if we set the argument to `[0 1]`, then the learning will be solely based on the discrete observation. The other inputs to this function are read directly from the data, except for `Valid`, which is set as we described above.

The function output is our learning state of interest. It returns forward-only filter (`XPos`, `SPos`) and forward-backward smoother (`XSmt`, `SSmt`) estimates of the state variables plus updated parameters. Besides that, we get the model likelihood at each iteration (`ML`) and the expected reaction time and probability of correct choice per trial (`YP`, `YB`). These are the core pieces of information that shape our inference about the data and underlying state variables. Figure 2 shows the filter estimate for the learning state with its confidence bound.

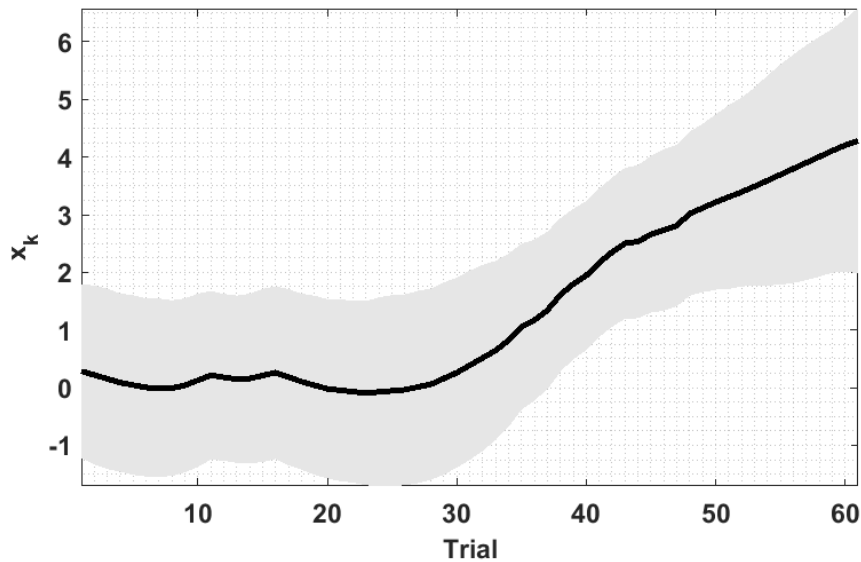


Figure 2 Posterior estimate of x_k . The c_0 is -1.3874; the sum of c_0 and x_k determines the probability of a correct choice per trial. The b_1 is 0.0479, which defines how the reaction time increases over trials. The x_1 is about 4 at trial 60, suggesting that the reaction time is about 20% higher as the animal proceeds through a block.

Figure 3 shows the ML estimate over 250 trials. We generally expect an increasing ML curve over trials, although this might not be always true depending on the approximations included in the model and choice of initial value of the model. To increase the likelihood of increasing ML (convergence), we can set initial values of the model parameters. These can be derived using a static generalized linear model (GLM) or similar analysis. Initial values for the model parameters are:

```
Param.Wk = 0.2;
Param.Ck = 0.1;
Param.Dk = [0 -1.66];
Param.Vk = 0.05;
Param.Fk = [0 -.15];
```

Default values of these parameters are provided in `compass_create_state_space`; however, these setting might not be optimal for all data sets. Here, we describe how better initial values can be derived for these parameters and for modeling analyses generally.

We can see that the animal learns the task as trials go by, and that this corresponds to an increasing x_k over trials. Note that the probability of correct choice increases as x_k increases. On the other hand, we set the initial value of x_0 to zero. Thus, we expect x_k to be a positive variable specifically at the end of the experiment. On the other hand, we observe that most of the choices are incorrect at the beginning of the task. Thus, this suggests c_0 , which is `Param.Fk(2)`, should be negative. In this experiment, there are 19 correct responses out of 39 trials (with a response). We can thus set the initial values of c_0 with a number that represents $19/39 \sim 48\%$ accuracy. Thus, we set `Param.Fk(2)` to $-0.15 - \log\left(\frac{p}{1-p}\right)$, $p = 0.462$ which is about this percentage of correct response. You can check `Param` definition to find how `Param.Fk(2)` relates to c_0 . We can take a similar rationale for the reaction time model. Mean of Y_n , the reaction time, is -1.66 , and thus we can set b_0 , which is `Param.Dk(2)`, to this number. Y_n variance is about 0.05 , and thus, we can set `Param.Vk` to this number. We know x_k grows over trials, and so does reaction time. This suggests that b_1 , which is `Param.Ck`, should be a positive number. We see from the choice data, that the animal responds with an accuracy of 90% or more, which corresponds to x_k of 2 or 3. Thus, we might assume x_k is 3 at the end of the task, and by checking reaction time at the end of the task, which is about -1.4 , we set b_1 to 0.1 . The last parameter is `Param.Wk`. We assume that variability of the behavior might have two sources. There can be variability in the response itself or in the underlying state (learning state). We can see the assumed variability in reaction time is about 0.05 , and there might be another variability which is generated by x_k . Note that we set b_1 to a small number, 0.1 here, which suggests x_k variance should be larger in order to generate reaction time variability. We can also assume that the variability of x_k be large enough that the E-M models allows it to change over trials. As a result, we keep x_k 's variance larger than `Param.Vk` and set to 0.2 , `Param.Wk` = 0.2 . Note that these are not necessarily the only choice for these parameters. The reasoning here might be useful, however, in defining other behavioral models.

Figure 3 shows the ML curve over 250 iterations. The curve reaches a plateau, which corresponds to a local maximum. Note that, if we start with another initial setting different from the setting used here, we might converge to another local maximum. In general, we are interested in a set of parameters and state estimation which lead to a meaningful description for the model we have built. For instance, we build the model by the assumption that learning state (x_k) increases over the experiment and we can see that the model output is aligned with our hypothesis.

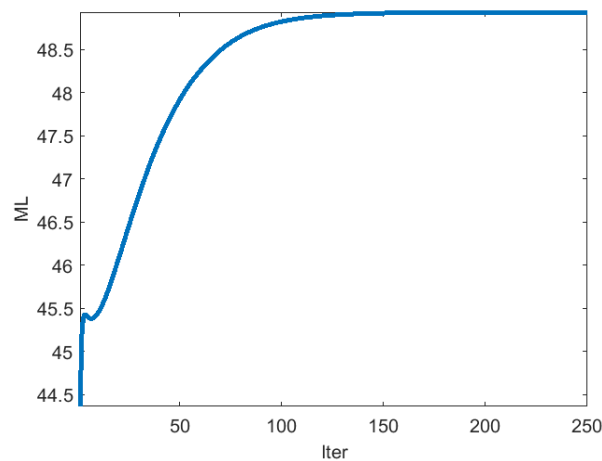


Figure 3 ML curve over 250 trials. The curve reaches a plateau which corresponds to a local maximum.

By running `compass_em`, we get the posterior estimate of the state variable ($XPos$, $SPos$) and updated model parameters. We get an ML curve as shown in Figure 3. The final output is the expected reaction

time and probability of choice per trial (Y_P, Y_B). This can be used to check the model's fit to the data, in terms of whether it is able to predict the observed output. Note that in the example, we have MAR points and we estimate expected the reaction time and probability of correct choice on those trials too. Figure 4 show the expected reaction time and probability of correct choice per trial.

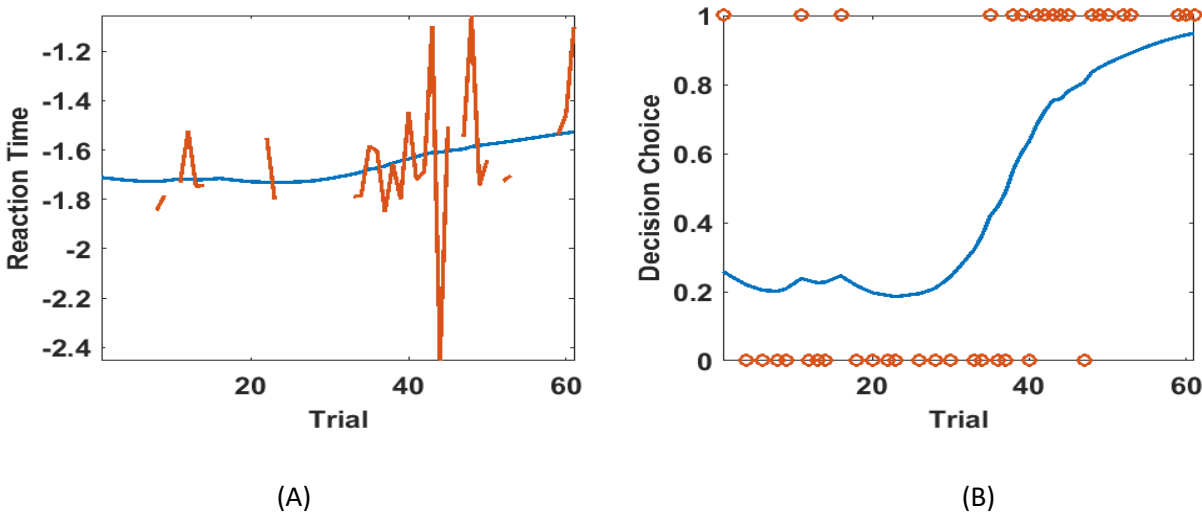


Figure 4 Expected reaction time (Panel A, blue) and expected probability of correct choice (Panel B, blue). Note that because the state evolves smoothly through time, we have estimated reaction times and choices even on trials where the subject did not respond. (A) The predicted reaction time (blue) follows the central tendency of the observations (orange), effectively acting as a smoothed version of the reaction time. In Figure 2, we can see that the confidence interval of x_k grows at the end of experiment along with the x_k growth, and this might reflect the larger variability in the reaction time at the end of the experiment. (B) The decision prediction (blue) suggests that probability of correct choice grows from around 20% to above 90%. The dot points (Panel B, orange) show correct/incorrect choice per each trial. A value of zero means an incorrect choice and one means a correct choice being made on that specific trial.

If we are considering multiple models, we can take further steps to compare models or assess the parameters estimation. For instance, we can call `compass_deviance` to estimate deviance over the continuous and discrete observation processes. Deviance is a goodness-of-fit statistic for a statistical model. It is a generalization of the idea of using the sum of squares of residuals in ordinary least squares fitting problems (Reference 4 – Ch2; Reference 5 -Ch7).

%% Deviance analysis

```
[DEV_C,DEV_D]= compass_deviance([1 1],In,Ib,Yn,Yb,Param,Valid,XSmt,SSmt);
```

The inputs to this function are the outputs and/or inputs of `compass_em`. We can use the deviance result to assess goodness-of-fit across different models (Reference 5 -Ch7). For instance, in modeling the learning behavior, we might assume that the reaction time (RT) is dependent on the trial number. Whether this component is related to RT can be checked by the changes in the deviance – in other words, changes in the deviance carry information about the significance of this new component in predicting RT. Naively speaking, a smaller deviance corresponds to a better fit to the data. Note that in the deviance measurement, there is no penalty term for the number of free parameters or the dimension of the state

variable. That type of penalty term is often used to avoid over-fitting a model to data (Reference 6). Let's assume we want to compare two other models to the model we built so far. First, we consider a model where the correct/incorrect choice depends only on a fixed parameter c_0 that is unaffected by learning:

$$p(y_{b,k} = 1) = \exp(c_0) / (1 + \exp(c_0)) \quad (7)$$

There are still both continuous and discrete observations, but only the continuous observation is now linked to the state. Model parameters, including c_0 , are learned from data. The code for this model is

```
Param.Wk = 0.2;
Param.Ck = 0.1;
Param.Dk = [0 -1.66];
Param.Vk = 0.05;
Param.Ek = 0;
Param.Fk = [0 -.15];
[XSmt, SSmt, Param, XPos, SPos, ML, YP, YB] =
    compass_em([1 1], Uk, In, Ib, Yn, Yb, Param, Valid);

[DEV_C1, DEV_D1] =
    compass_deviance([1 1], In, Ib, Yn, Yb, Param, Valid, XSmt, SSmt);
```

For the second alternate model, we assume we only have the binary observation. This means the state estimation and model parameter estimation are only derived from the binary observation and the reaction time is not considered. The code implementation is

```
Param.Wk = 0.2;
Param.Ck = 0.1;
Param.Dk = [0 -1.66];
Param.Vk = 0.05;
Param.Ek = 1;
Param.Fk = [0 -.15];
[XSmt, SSmt, Param, XPos, SPos, ML, YP, YB] =
    compass_em([0 1], Uk, In, Ib, Yn, Yb, Param, Valid);
[DEV_C2, DEV_D2] =
    compass_deviance([1 1], In, Ib, Yn, Yb, Param, Valid, XSmt, SSmt);
```

We can see the deviance analysis for these three models (the original and two alternative models) in the following table.

Table 1 Deviance Comparison Between Different Models

Model	Deviance (Continuous Obs.)	Deviance (Discrete Obs.)
Original Model	-7.7	36.9
Model with both observations, and without a link to the state in the binary observation	-3.4	54.0
Model with binary observation alone	32.8	38.5

The deviance result between the original model and the first alternative model suggests that including the state variable in the discrete model will improve our data explanation. This is because we see a big increase in the deviance result on the discrete part. The result also suggests that we might get a better explanation of reaction time in the first alternative model; however, `Param.Ck (=0.046)` becomes tiny suggesting that the reaction time alone might not carry much information about learning state. In the second alternative method, the deviance has increased a bit for the discrete observation, suggesting that the binary observation alone carries less information about the learning state compared to when both reaction time and choices are utilized in the model. This is in accordance with the original model result. Even though the reaction time on its own does not carry much information about learning, it does carry information when specifically conditioned on/considered in parallel with the subject's choices. . Note that, in the second alternative model, we don't tune the continuous observation parameters and thus we get a large deviance. This sort of analysis provides a view about which observation carries more information and how we can tune our model structure. We could further examine other outputs of the alternative models, such as the time course of the estimated learning state and the static model parameters. Now, we will discuss the covariance estimate of the model coefficients which provides an extra information about the model structure.

Another key model verification analysis is the covariance estimate of the model free parameters. For instance, we might be interested in checking the confidence interval for c_0 , b_0 or b_1 estimates or the model noise parameter estimation. To find this information, we can call the following function:

```
%% Covariance analysis
```

```
[COV_X, COV_C, COV_D] = compass_param_covariance_info([1
1], Uk, In, Ib, Yn, Yb, Param, Valid, XSmt, SSmt);
```

Figure 5 shows the confidence interval for the model parameters of our original example model. The result is in accordance with deviance analysis; we can see that b_1 is close to 0, with a confidence interval almost including 0 –we discussed that RT is less dependent to learning state and decision carries much of information about the learning state . This implies that the reaction time is relatively insensitive to the state. We can see a larger confidence interval for c_0 . The state carries much of the information about the choice decision, and this level of information changes over time. This does suggest that we might want more data to narrow this coefficient estimate. We see a larger variance estimate for the state process σ_w^2 , shown by Sw , compared to the variance of the observation process σ_v^2 , shown by Sv .

These two functions along with `compass_em` output provide a reasonably inclusive picture of the model and state estimation. We can use these information for refining the model structure similar to what is done in static GLM selection (Reference 5- Ch7). This will be similar to a model selection process, when new covariates are added or dropped sequentially. A cautionary point is that no single number or output from these analyses indicates that one model is preferable. Rather, the model is selected based on how well the analysis matches what we know or assume about the problem – our pre-existing hypotheses or domain-specific knowledge.

In this example, we worked over different steps that can be done in COMPASS to build a model and analyze its result. We can use COMPASS to build different models and examine how they might explain our signal.

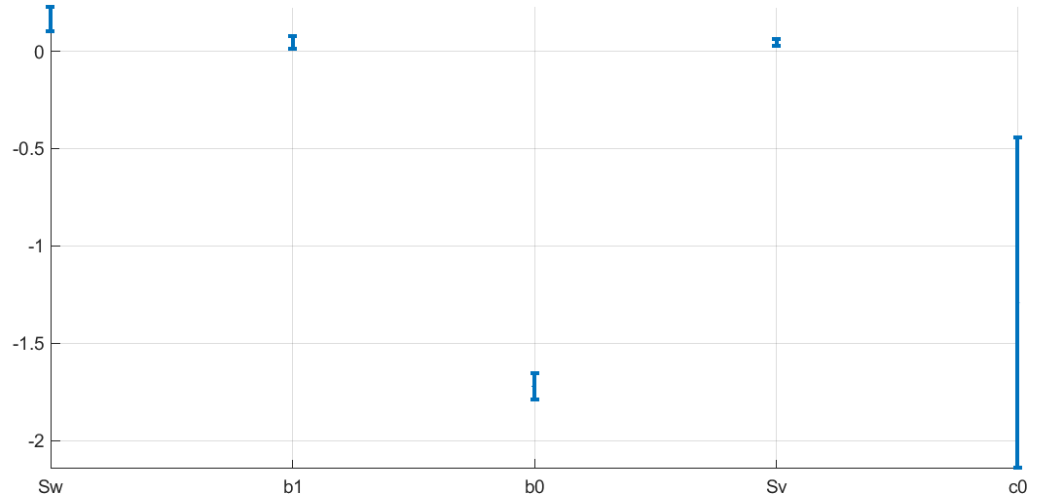


Figure 5 Confidence interval of the model free parameters. The result suggests that b_1 might be close to zero, which suggests the reaction time is less dependent to the learning state (x_k).

Model Creation Wrapper Function: [compass_create_model.m](#)

[compass_create_model](#) is a wrapper function combining [compass_create_state_space](#), [compass_set_learning_param](#), and [compass_set_censor_threshold_proc_mode](#) to create a set of input arguments to be passed through the [compass_em](#) function. Using this function, the user can create a state-space model and set learning and threshold parameters easily by utilizing a series of question and answer functions. This allows the user to easily construct a desired model without possessing an in-depth understanding of each toolbox function that is necessary for model construction. The function itself is versatile and will adapt to various user preferences and data types when choosing a model structure, and will furthermore allow the user to run an unlimited amount of models consecutively through [compass_em](#) without interruption.

Upon initially running this function, the user will be prompted to enter the path and name of a data file containing the following variables. The function assumes that the user has already created this file in mat format, and that the user knows where it is located. If no path is given, the file must be located in the same directory as the function in order to proceed.

Yn – Vector of continuous observation data signal, e.g. reaction time. Vector size: $K \times 1$, where K is the number of observed data points.

Yb – Vector of discrete observation data signal, e.g. correct/incorrect response. Vector size: $K \times 1$.

In – Matrix of continuous observation input. Matrix size: $K \times nIn$, where nIn is a scalar defining the dimension of input passed to the continuous part of the model.

Ib – Matrix of discrete observation input. Matrix size: $K \times nIb$, where nIb is a scalar defining the dimension of input passed to the discrete part of the model.

obs_valid – Vector that determines whether each data point is completely observed (with a value of 1), missing at random (with a value of 0), or censored by threshold ([with a value 2](#)). Vector size: $K \times 1$. If this variable is not provided in the user inputted data file, it will be created as a vector of dimensions $K \times 1$. All values will be equal to “1.”

Uk – Matrix that defines input to state-transition model across time indices. If this variable is not given, it will be created as an empty vector: [].

NOTE: The bare minimum variables needed to run this function are a pair of either **Yb** and **Yn**, or **In** and **Ib**. This will allow for processing of continuous variables and/or discrete variables respectively. If neither of these two sets of variables are within the uploaded data file, the function will terminate with the message “*Error: Function cannot be executed due to lack of present variables* Check data file for necessary variables and proper spelling/capitalization.”. The function will account for missing discrete variables as long as there are present continuous variables, and vice versa. Proper model construction for the presence/absence of these variables will be accounted for by the function itself.

Once all model-based questions are answered, the user will finally be prompted to save their new input arguments in a mat-file. The user will be prompted to add a suffix to the given prefixed file-name “em_set.” Then, they will be capable of running **compass_em** easily with these new parameters using the function **compass_run_models** (described below). Using **compass_run_models** in accordance with **compass_create_model** will allow users to run multiple model settings at a time without interruption. After each variable input set is passed using **compass_run_models**, the resulting model information will be saved as a new mat file named “model_result” with an added suffix number given by the order in which the variable input set was received by the function. This process will be explained further below at the end of the example problem.

In the following sections, we describe each step of this function by building a particular model including both continuous and discrete observations using **compass_create_model**. This will include an entire run through of the function, spanning question and answer functions regarding **compass_create_state_space**, **compass_set_learning_param**, and **compass_set_censor_threshold_proc_mode**. It should be noted that the nature of questions presented by **compass_create_model** will differ pending the data uploaded by the user, and the answers that the user provides in question and answer functions. This is merely one example of many potential models that can be generated using this function.

Example Model Using **compass_create_model.m** Wrapper function

Let’s assume the user would like to model the following equations:

$$Y_n(k) \sim f((2 I_{n,1}(k) + \beta_1 I_{n,2})x_1(k) + I_{n,3}(k)\beta_3; \theta)$$

$$Y_b(k) \sim g(I_{b,1}(k)x_1(k)w_1 + I_{b,2}(k)x_2(k)w_2; \theta)$$

In the above equations, Y_n stands for a continuous data signal and Y_b stands for a discrete data signal. I_n and I_b stand for input of continuous and discrete observations respectively. $x_1(k)$ and $x_2(k)$ represent model state variables, while β and w represent model parameters. θ represents remainder parameters of the model, including noise terms. Note that for the continuous model, we can assume a different distribution model, which will be one of the questions being called in the function. For discrete observation, we assume $Y_b(k)$ follows a Bernoulli distribution, thus $Y_b(k)$ will have a value of 0 or 1 at each time index. Note that, in the continuous model, the coefficient of the first term is fixed – equal to 2 – for $I_{n,1}(k)$ while other parameters described (β_1, w_1, w_2) are assumed to be free. The toolbox has the capability of handling this and it will be described below through the questions being asked by the function. Below, it is assumed that a mat file containing **Yn**, **Yb**, **In**, **Ib**, **obs_valid**, and **Uk** is being loaded. The following answers will generate these two equations, as well as prepare a series of input arguments necessary to be passed through **compass_em**:

1. Please provide the exact path to your mat variable input file:

Here, the user will provide the exact path and name of their mat variable input arguments.

Ex. G:\MATLAB\COMPASS_StateSpaceToolbox\example_data.mat

2. Do you have continuous observations? (Y/N)

“N” would exclude continuous observations from the model, which involves variables I_n and Y_n .

Answer “Y” because there is an equation involving continuous observation (Y_n).

3. Do you have discrete observations? (Y/N)

Answer “Y” because there is an equation involving discrete observation (Y_b).

4. Which distribution for continuous observations? (If answered “Y” to continuous observations)

Picking 2 would result in your model following a Gamma distribution.

Pick 1 to have your model follow a normal distribution.

5. How many state variables do you have?

Answer “2” because there are two state variables, $x_1(k)$ and $x_2(k)$.

CONTINUOUS STATE VARIABLES

6. Which state variable will be linked to column number “1” in your I_n matrix?

Answering “0” will indicate that no state variable will be linked to this column.

Answer “1” because state variable $x_1(k)$ is linked to column number 1 in the I_n matrix (I_n), following the above equation

7. Is this a free or fixed parameter? ($I_n(1) * b * x_1$) Enter 1 for free, 0 for fixed:

Answer “0” for fixed because there is no changing free parameter associated with the $I_{n,1}$ term.

8. What is the initial value of the parameter?

Answer “2” because the $I_{n,1}$ term has a coefficient of 2.

9. Which state variable will be linked to column number “2” in your I_n matrix?

Answer “1” because $x_1(k)$ is linked to column number 2 in the I_n matrix (I_n).

10. Is this a free or fixed parameter? ($I_n(2) * b * x_1$) Enter 1 for free, 0 for fixed:

Answer “1” for fixed because there is a changing β_2 variable associated with the $I_{n,2}$ term.

11. What is the initial value of the parameter?

This value is up to the user, let’s say it is around 1.2.

12. Which state variable will be linked to column number “3” in your I_n matrix?

Answer “0” because there is no state variable linked to column number 3 in the I_n matrix (I_n).

13. Is this a free or fixed parameter? ($\ln(3) \cdot b$) Enter 1 for free, 0 for fixed:

Answer "1" for free because there is a changing β_2 variable associated with the $I_{n,3}$ term.

14. What is the initial value of the parameter?

This value is up to the user, let's say it is around 1.4

CONTINUOUS MODEL: $Y_n \sim f(\ln(1) \cdot 2 \cdot x_1) \cdot (\ln(2) \cdot (b_1 -> 1.2) \cdot x_1) \cdot (\ln(3) \cdot b_2 -> 1.4)$

Press enter to continue

DISCRETE STATE VARIABLES

15. Which state variable will be linked to column number "1" in your I_b matrix?

Answer "1" because $x_1(k)$ is linked to column number 1 in the I_b matrix (I_b).

16. Is this a free or fixed parameter? ($I_b(1) \cdot b \cdot x_1$) Enter 1 for free, 0 for fixed:

Answer "1" for free because there is a changing w_1 variable associated with the $I_{b,1}$ term.

17. What is the initial value of the parameter?

This value is up to the user, let's say it is around 1.2.

18. Which state variable will be linked to column number "2" in your I_b matrix?

Answer "2" because $x_2(k)$ is linked to column number 2 in the I_b matrix (I_b).

19. Is this a free or fixed parameter? ($I_b(1) \cdot b \cdot x_1$) Enter 1 for free, 0 for fixed:

Answer "1" for free because there is a changing w_2 variable associated with the $I_{b,2}$ term

20. What is the initial value of the parameter?

This value is up to the user, let's say it is around 1.4.

DISCRETE MODEL: $Y_b \sim g(I_b(1) \cdot (w_1 -> 1.2) \cdot x_1) \cdot (I_b(2) \cdot (w_2 -> 1.4) \cdot x_2)$

Press enter to continue

Learning Parameters Update

Enter "1" for "yes", or "2" for "no" *UNLESS OTHERWISE SPECIFIED*

21. Update state-transition model parameters?

A value of 1 updates elements of **A** and **B** matrices, whereas a value of 0 will not update these matrices.

Let's choose a value of 1.

22. Update state-transition model covariance matrix?

A value of 1 updates **W**, the covariance matrix. A value of 0 will leave the **W** matrix un-updated.

Let's choose a value of 1.

23. Update initial state variable parameters?

A value of 1 updates state variable parameters, the mean and covariance matrices, at time 0. A value of 0 will leave these parameters un-updated, and the user can manually initialize variables **Param.X0** and **Param.W0**.

Let's choose a value of 1.

24. Update continuous parameters?

A value of 1 will update the continuous observation process during EM training. A value of 0 will leave these parameters un-updated. If continuous observations are not present, this question will be skipped.

Continuous observations are present, so we can choose a value of 1.

25. Update noise-term?

A value of 1 will update the noise-term during the EM training process. In continuous observation processes with a normal or Gamma distribution, the noise term represents additive normal white noise. Whereas in continuous processes with a Gamma distribution, the noise term corresponds to a dispersion term. If continuous observation is not present, this question will be skipped.

Continuous observation is present, so we can choose a value of 1.

26. Update discrete parameters?

This question and answer choices are identical to question #24 (update the observation process during EM training) but for the discrete observation process. If discrete observations are not present, this question will be skipped.

Discrete observations are present, so we can choose a value of 1.

27. Is matrix "A" in state-transition process diagonal or not?

A value of 1 will make the A matrix in the state transition process model diagonal, which is the favorable choice for EM learning. A value of 0 will make the A matrix a full matrix, which could lead to less accurate or unstable estimations of both parameters and state-variables.

Let's choose a value of 1.

28. Update mean before covariance (ENTER 1), or covariance before mean (ENTER 2)?

For the filter mean and covariance update, this determines if mean is updated before covariance (a value of 1), or vice versa (a value of 2). Further information about this topic can be found in Reference 3.

Let's choose a value of 1.

29. Update a positive shift estimation during the training process?

A value of 1 sets the estimation of a positive shift as part of the EM training process, updating this estimation during training. A value of zero will make this shift parameter pre-set and un-updated in the EM training process. This is applicable to the Gamma distribution only, so let's choose a value of 0.

Learning parameters are set.

Threshold Parameters Update

30. Would you like to include a threshold in your model?(Y/N)

It should be noted that the user will only be asked this and the next three questions if there is at least one "2" value present in the **obs_valid** vector of their initial variable set. Answering "Y" will set a censor threshold to be utilized on the continuous observation process during training. Thus, any time index with a numerical value of "2" within the **obs_valid** vector will be treated as a censored data point. If **obs_valid** is not provided by the user's initial variable set, the function will create an **obs_valid** variable in which all values are set to "1."

Let's choose "Y" assuming we have at least one "2" in our **obs_valid** vector.

31. Please enter a numerical value for your threshold.

Trials exceeding this given value will be censored.

This number is up to the user; let's answer a value around 1.3.

32. Please enter 1 to process censored data with imputation, or enter 2 to process censored data with a Gaussian approximation.

A value of 1 means imputation method will be used for modeling censored data points. A value 2 means a full likelihood will be used in estimating the state variables. For further discussion, please check reference 3.

Let's choose a value of 1.

33. Enter 1 to update filter mean before covariance, or enter 2 to update covariance before filter mean.

A value of 1 indicates that the mean is updated before covariance during the EM process, and vice versa for a value of 2. For further discussion, please check reference 3.

Let's choose a value of 1.

Threshold parameters are set.

34. Enter the number of compass_em training iterations.

This numerical answer will determine how many iterations **compass_em** will run through during training. Let's answer "250."

35. Please enter the full path of where to save your new variable set, ending with a backslash ("\"):

Here the user must enter the entire path of where to save the new variable set. The user must not forget to add a backslash at the end of the path. This step can be skipped by pressing "enter", and the file will be instead saved in the current working directory.

Ex. G:\MATLAB\COMPASS_StateSpaceToolbox\

36. Please enter the suffix you would like to add to the "em_set" filename:

Here the user may enter any suffix they would like. The user must be aware that the file will be saved with the prefix "**em_set**" and not merely as the suffix they present alone. Let's input the number "1" which will result in a saved mat file named **em_set1.mat**.

Once all questions are answered, **compass_create_model** will save an updated set of variable inputs to be passed through **compass_em**.

Wrapper Function **compass_run_models.m to Run Multiple Models**

compass_run_models will automatically pass variable input sets one by one through **compass_em** and save their corresponding outputs in the StateSpaceToolbox folder of the user's computer. Each set of output data will be saved as a mat file, labeled **model_result** preceding a suffix number corresponding to the order that the file was presented (in list-form) to the function. **compass_run_models** allows users to upload as many mat variable files as they so choose, thus generating as many updated models as they would like.

NOTE: Input to **compass_run_models** **must** be passed as strings in list form. In other words, the input to this function must be surrounded by curly brackets, and within the curly brackets mat files must be presented as strings (thus each file must be surrounded by single apostrophes and separated by commas).

Ex. The user would like to pass three input variable sets through **compass_em**. These variable sets are named **em_set1.mat**, **em_set2.mat**, and **em_set3.mat** (this follows the naming framework of **compass_run_models**).

The user will simply pass the names of each input variable sets through **compass_run_models**, and the resulting output data will be automatically saved as **model_result1.mat**, **model_result2.mat**, and **model_result3.mat**. Described in one line of code below, passing the input variable sets will result in the aforementioned saved output data sets:

```
compass_run_models({'em_set1.mat','em_set2.mat','em_set3.mat'})
```

NOTE: the order in which the variable sets are passed through `compass_run_models` will determine the naming suffixes of the resulting output variable files. Thus, if the aforementioned variable sets were presented in reverse order like so:

```
compass_run_models({'em_set3.mat','em_set2.mat','em_set1.mat'})
```

The output of `em_set3` would be named `model_result1`, of `em_set2` would be named `model_result 2`, and of `em_set1` would be named `model_result1`.

Running Multiple Model Using `compass_run_models.m`

Using `compass_create_models` we will create three models. The data files in these examples are derived from the `mat` file called `LEARNING_1.mat` which can be found in the [GitHub](#) repository. This `mat` file includes the behavioral outcome per trial, including both reaction time and binary (correct/incorrect) decisions. The data come from an associative learning task, where through the task, a rhesus macaque learns the correct response on each trial to receive reward.

The first model in this example is mimicked from the first example of the example problems located on page 1 of the manual. It will include both continuous and discrete observation, with `ln` and `lb` each including one singular column. All non-response trials will be considered missing at random, thus their corresponding `obs_valid` vector locations will all be set equal to zero. There will be only one state variable, with `nx` set equal to one.

The second model is exactly the same as the first, however it uses continuous observation exclusively and does not include discrete observation. This change will be encountered for by the user through the use of `compass_create_models`.

The third model is equivalent to the first model, however it includes the use of a threshold and involves the function `compass_set_censor_threshold_proc_mode`. Thus, all non-response trials will now be considered as censored data points, and their corresponding locations in `obs_valid` will be set equal to 2.

Further instruction on how to create these three models using `compass_create_models` can be found in your COMPASS toolbox folder, in the script named "`compass_example_E.m`", under the first, second, and third headers labeled %% Model #1, %% Model #2, and %% Model #3. Here, step by step directions will be given for the creation of these models using `compass_create_models`.

Once all three models are constructed and their respective input variable sets are saved, `compass_run_models` will be called and three corresponding `compass_em` output sets will be generated and saved as individual `mat` files. This function can be found in your COMPASS_StateSpaceToolbox folder.

Censored Data Problem Example

In this example, we study a state and parameter estimation problem in a dataset with censored data points. We generate the state and observation data using the following model:

$$x_{k+1} = a x_k + v_k \quad v_k \sim N(0, s_v^2)$$

$$y_k = x_k + w_k \quad w_k \sim N(0, s_w^2)$$

We define a set of threshold values – T_i – which defines the upper bound on observation. Given the threshold, the observation is defined by

$$o_k = \begin{cases} y_k & y_k < T_i \\ \text{censored} & y_k \geq T_i \end{cases}$$

The objective is to estimate model parameters – (a, s_v^2, s_w^2) – and state – x_k – given o_k and T_i .

We simulate the data using the following parameter settings:

$$a = 0.99$$

$$s_v^2 = 0.01$$

$$s_w^2 = 0.055$$

and we assume the length of observation is 200 – $k = 1, \dots, 200$. Below figure shows the simulated data – x_k, y_k, o_k . For o_k , the threshold – T_i – is set to 0.1563, from a possible set of values ranging from -0.6131 to 1.2143.

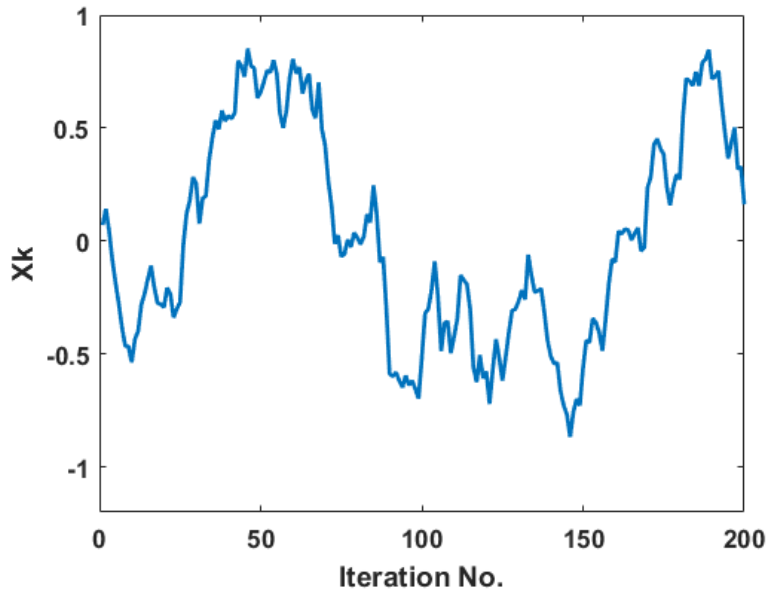


Figure 8 x_k state estimate

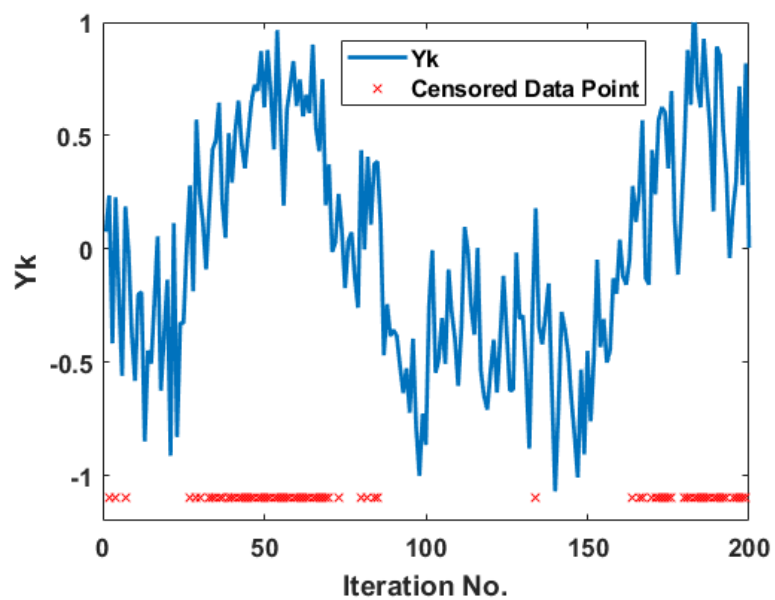


Figure 9 y_k true value

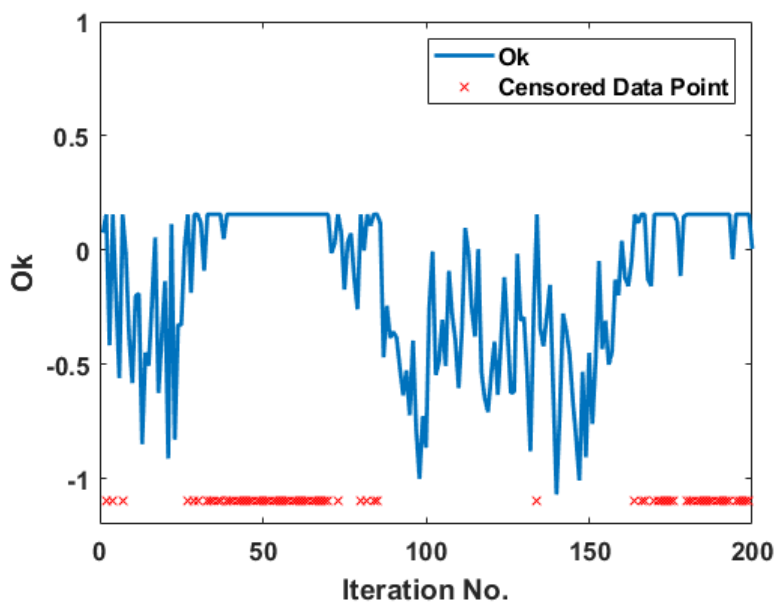


Figure 10 o_k with $T_i = 0.1563$

We use [compass_create_example_F_data.m](#) to generate the data. The Matlab code for generating this data is as follows, and the simulation result is save in the [em_set100.mat](#) file.

```
%% Set Initial Parameters
sx = 0.01;
sy = 0.055;
K = 200;
a=0.99;
x0 = sqrt(sx)*randn();
%Create estimate of Yn
for k=1:K
    if k==1
        x(k)=a*x0 + sqrt(sx)*randn();
    else
        x(k)=a*x(k-1) + sqrt(sx)*randn();
    end
end
%Create Yn
for k=1:K
    y(k)= x(k) + sqrt(sy)*randn();
end

%% RMSE
max_y = 0.1+max(y);
min_y = min(y)+0.2*(max_y-min(y));
th_s = linspace(min_y,max_y,20);
save('example_var','th_s','sx','sy','x','y')%save th_s, x, sx, sy

%% Set Model Parameters
In = ones(K,1);
Yn = y';
obs_valid=ones(K,1)*2;
save('model4_variables','In','Yn','obs_valid')

%% Create Original Model
compass_create_model

load('em_set100')
Param.Ak = 0.95;
save('em_set100')
```

We use COMPASS to estimate the state and model parameters. COMPASS can handle the censored data problem three different ways. In the first method, the censored data are assumed to be randomly dropped – MAR – which is generally the naïve and less accurate model as no information is being passed on the censored data points. The second method is imputation, where COMPASS generates a sample per each censored data point. The last method is the full likelihood method, where COMPASS uses the full likelihood of censored data points in both state and parameter estimation. In this example, we study MAR and full-likelihood method results for a different set of thresholds. The assumption is the full likelihood method is more robust and can estimate the state and parameters with a better accuracy even in presence of many censored data points. To assess COMPASS' and different algorithms' performance in estimating state, we use RMSE between estimated state mean and true state generated by simulation. For parameter estimation, we visualize the deviation of parameters from their ground truth values. The hypothesis is RMSE stays reasonably the same independent of the number of censored data points in the full-likelihood method, whereas it grows in the MAR method as the number of censored data points increase. Similarly, we assume parameter estimation in the full-likelihood method is more accurate and robust in the presence of censored data points. Note that when there are no censored data points, MAR and full likelihood have the same performance.

We use the `compass_create_example_F_data.m` file to create the original model, and we then build more models using the `compass_run_models` function. Note that the structure of COMPASS models is the same for both MAR and full-likelihood methods; however, they need different settings to run different estimation methods. For MAR, we only need to change the **obs_valid** variable: All values exceeding the given threshold are set equal to zero. For the full likelihood method, we need two changes. First, in **obs_valid**, all values exceeding the given threshold are set equal to two. The second change is in Param. For each threshold value, we need to pass the proper value to this function, which is defined by setting Param.censor_time equal to the desired threshold value.

Within `compass_example_F.m`, the changes needed to account for both MAR and full-likelihood models are made. The code iterates through each threshold value. For the full-likelihood model, the code changed obs_valid elements to two if corresponding **Yn** values are greater than the threshold. For the MAR model, these corresponding elements are changed to zero. Once obs_valid is altered, for the full-likelihood model, the resulting data is saved into a new data set labeled 'emset1' with a suffix ranging from 1-20 (20 is the number of threshold values), pending on the order in which the data was created. For the MAR model, the resulting data is saved into a new data set labeled 'emset2' with the same suffix system as for the full-likelihood model.

Once 20 data sets are each saved for the full-likelihood and MAR cases (40 sets total), `compass_run_models` iterates through all 40 sets, passing these datasets through compass_em one by one. As per the functioning of `compass_run_models`, the resulting parameters from each set will be saved as "model_result" with a suffix equal to the numbered order in which the original dataset was passed through the function.

Below, Figure 9A shows the number of censored data points for different T_i . T_i changes from -0.6131 to 1.2143 with 20 linearly spaced samples; as the threshold drops the number of censored data points increases. Figures 9B-9C show RMSE plus parameter estimates using MAR and full likelihood method for all given threshold levels. Figure 9B shows RMSE as function of censored threshold for both methods. The result is in accordance with our assumption, and it suggests the full likelihood method gives a more robust

estimation of the state. Figures 9C and 9D show parameter estimates using both methods for different threshold levels. Note that both methods provide an accurate estimation of the model parameters when there are no censored data points. As the number of censored data points increases, we expect the estimation of these parameters to be biased. Note that the full-likelihood method is still robust and provides an accurate estimation of model parameters even with a large number of censored data points.

Figure 9

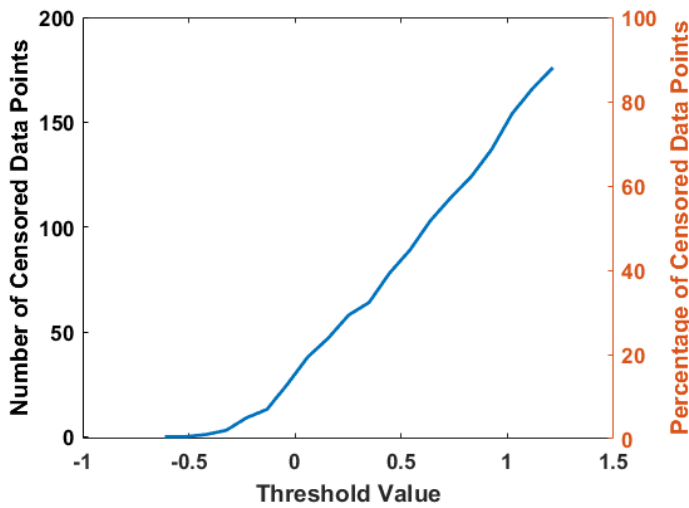


Figure 9A Number of censored data points increases linearly as threshold value increases, exemplifying a positive correlation. Ideally, censored data points should be around 60% or else.

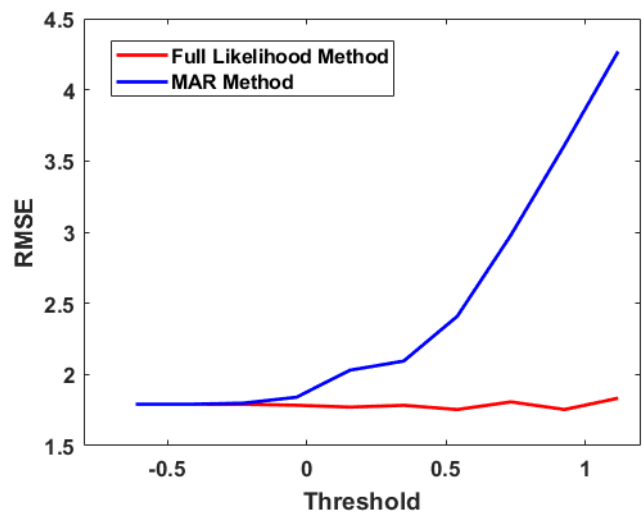


Figure 9B For full-likelihood estimate, RMSE curve remains fairly constant as threshold value increases. Unlike MAR estimate, where RMSE increases with threshold value.

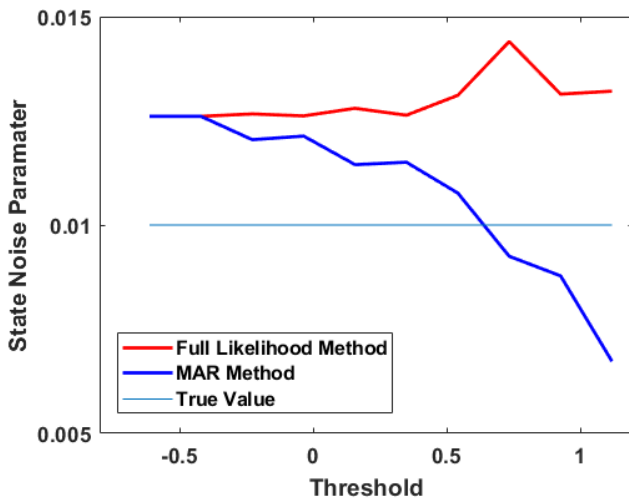


Figure 9C For state noise parameter estimate, full-likelihood estimate remains thoroughly consistent as threshold value increases. Both full-likelihood and MAR estimates are near true value.

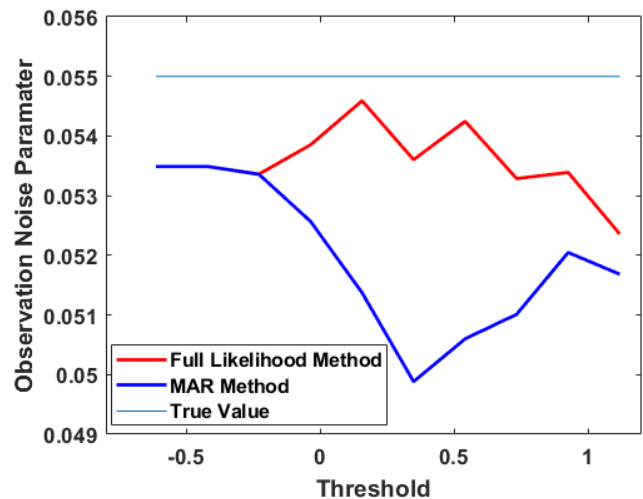


Figure 9D For observation noise parameter estimate, full-likelihood estimate undergoes slight fluctuations while remaining mostly constant. MAR estimate experiences trough and recovers.

Besides these result, we can study other components of COMPASS and model estimations. Figure 10 shows the state estimation and ML curve for T_i using Full likelihood and MAR models.

Figure 10

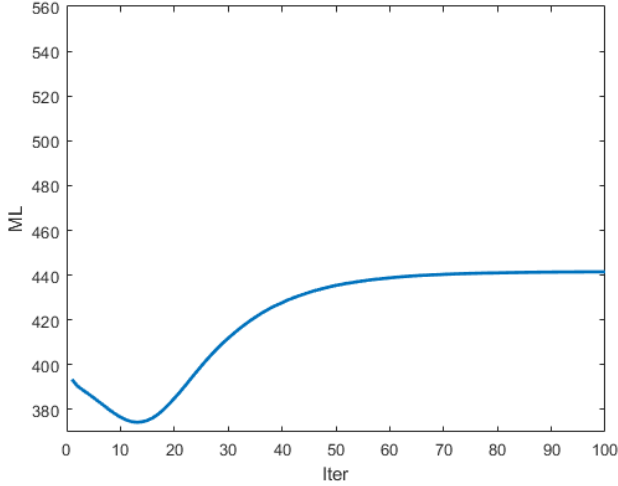


Figure 10A Full likelihood ML curve for $T_i = 0.1563$.

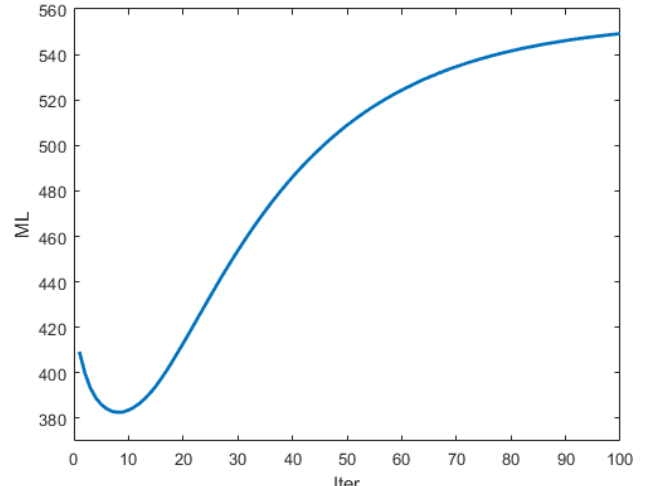


Figure 10B MAR ML curve for $T_i = 0.1563$.

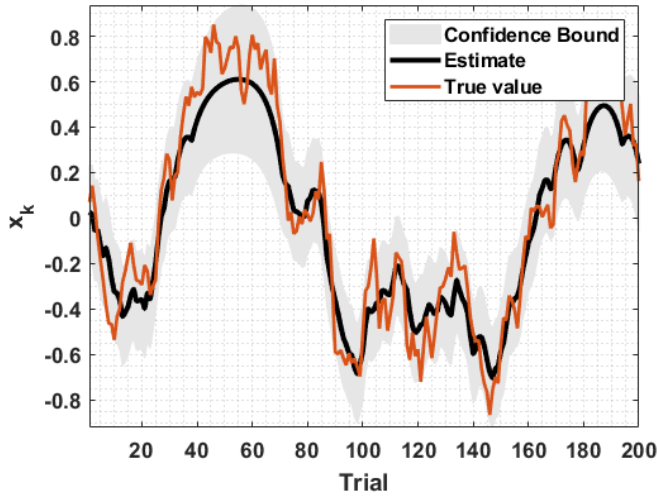


Figure 10C Full likelihood state estimate with confidence interval and actual value. Note the similarity between all three. $T_i = 0.1563$

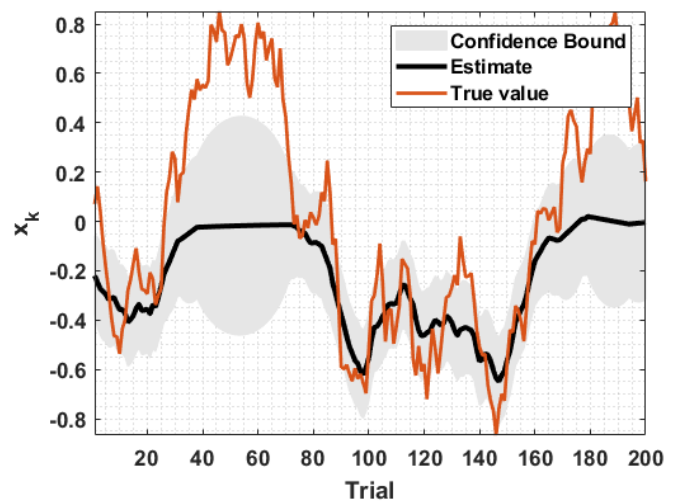


Figure 10D MAR state estimate with confidence interval and actual value. Note the difference between Figure 15. $T_i = 0.1563$

ML curves suggest that both models are converging to a local maximum, however they might provide different predictions of the state and model parameters. For the full likelihood model, the state estimation is reasonably accurate even on the censored data points, whereas for the MAR model, we can see the bias in estimation. Note that censored data points are providing information, and the full likelihood is using this information to predict model parameters and state.

What we have shown in this example shows COMPASS' capability in predicting the model parameters and state variables under different levels of censored data points. When there is not a censored data point, COMPASS properly estimates the model parameters and state variable. COMPASS is also capable of running different methodologies to estimate state and model parameters in presence of the censored data points and provide a robust and accurate estimation. This example provides a simple model, we can use COMPASS functions – as we described in [compass_create_example_F_data.m](#) and [compass_example_F.m](#) – to build more complex models and then use COMPASS utilities to predict model parameters.

Manual

Types of observed signals which can be passed to the toolbox are:

1. Normal/Log-Normal
2. Gamma
3. Bernoulli (discrete/binary choice)
4. Normal/Log-Normal + Bernoulli
5. Gamma + Bernoulli

Observed signals might include both continuous or discrete scalar values.

The toolbox includes the following functions:

1. [compass_create_state_space](#)
2. [compass_set_learning_param](#)
3. [compass_em](#)
4. [compass_filtering](#)
5. [compass_sampling](#)
6. [compass_set_censor_threshold_proc_mode](#)
7. [compass_deviance](#)
8. [compass_param_covariance_info](#)

Internal functions of this toolbox are:

1. [compass_Qk](#)
2. [compass_Tk](#)
3. [compass_x_2_rp](#)

Plot function [compass_plot_bound](#) can be used to plot state-variable confidence intervals.

References:

1. Yousefi, A., Paulk, A. C., Deckersbach, T., Dougherty, D. D., Eskandar, E. N., Widge, A. S., & Eden, U. T. (2015, August). Cognitive state prediction using an EM algorithm applied to gamma distributed data. In *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE* (pp. 7819-7824). IEEE..
2. Prerau, M. J., Smith, A. C., Eden, U. T., Kubota, Y., Yanike, M., Suzuki, W., ... & Brown, E. N. (2009). Characterizing learning by simultaneous analysis of continuous and binary measures of performance. *Journal of Neurophysiology*, 102(5), 3060-3072.
3. Yousefi, A., Dougherty, D. D., Eskandar, E. N., Widge, A. S., & Eden, U. T. (2017). Estimating Dynamic Signals From Trial Data With Censored Values. *Computational Psychiatry*, 1, 58-81.
4. Nelder, J. A., & Baker, R. J. (1972). *Generalized linear models*. John Wiley & Sons, Inc.

5. Venables, W.N. and Ripley, B.D., 2013. *Modern applied statistics with S-PLUS*. Springer Science & Business Media.
6. Kass, R.E., Eden, U.T. and Brown, E.N., 2014. *Analysis of neural data* (Vol. 491). New York: Springer.
7. Seltman, H.J., 2012. Experimental design and analysis. *Online at: <http://www.stat.cmu.edu/hselman/309/Book/Book.pdf>*.
8. Widge, A.S., Ellard, K.K., Paulk, A.C., Basu, I., Yousefi, A., Zorowitz, S., Gilmour, A., Afzal, A., Deckersbach, T., Cash, S.S. and Kramer, M.A., 2017. Treating refractory mental illness with closed-loop brain stimulation: Progress towards a patient-specific transdiagnostic approach. *Experimental neurology*, 287, pp.461-472.

Written by Ali Yousefi - Jan , 2018

compass_create_state_space

It creates the state space model for a specific task. The function defines how each continuous and discrete part of the model is defined, and how they are linked to the state variables. This is the first function required to be called in this toolbox.

Syntax

Param = compass_create_state_space(nx,nUk,nIn,nIb,xM,cLink,cLinkUpdate,dLink,dLinkUpdate)

Description

Param = compass_create_state_space(nx, nUk,nIn,nIb,xM,cLink,cLinkUpdate,dLink,dLinkUpdate)

The function returns the parameters of the specified behavioral model defined by the function input arguments. The **Param** contains the model structure and necessary parameters.

Input Arguments

nx	<p>A scalar that defines dimension of state variable (X)</p> <p>Example: $nx = 2$, this means X is a vector with dimension 2 - $X = [x_1 \ x_2]'$.</p>
nUk	<p>A scalar that defines the dimension of input to the state-space model</p> <p>$X(k + 1) = A X(k) + B U(k) + W$, $U(k)$ is the input vector with “nUk” elements.</p> <p>Example: $nUk = 2$, for instance the $U(k) = [1 \ s(k)]$ has a dimension of 2. The “1, $s(k)$” are the components of the input to the state transition model. For example, at time $k = 1$, $U(1) = [1 \ 0]$, which means $s(1)$ is equal to 0 at time 1.</p> <p>Note that nUk can be set to zero when there is no $U(k)$ term in the model.</p>
nIn	<p>A scalar that defines the dimension of input passed to the continuous part of the model $y_c(k) = g_c(I_n(k), X(k), P, v)$; $I_n(k)$ is the input vector with “nIn” columns and P are the model free parameters.</p> <p>Note that I_n will be a matrix with size $K \times nIn$, where K is the length of the process (the number of trials).</p> <p>For the Normal distribution, the $g_c()$ function is a linear function of the X and other model parameters:</p> $y_c(k) = CT_k X(k) + DT_k + v$ <p>Note that CT_k and DT_k are functions of $I_n(k)$ and P, and variable v is the observation process noise. The format of CT_k and DT_k is defined by the other input arguments (In, CLink, xM) to the <code>compass_create_state_space</code> function. Check xM, CLink and CLinkUpdate for more explanation.</p> <p>For the Gamma distribution, you might check the function definition described in Reference 1.</p>

	<p>Example: $nIn = 5$, for instance $I_n(k) = [I_b(k) \ I_c(k) \ I_h(k) \ I_{i2c}(k) \ I_{c2i}(k)]$ has a dimension of 5. The “I_b, I_c, I_h, I_{i2c}, and I_{c2i}” can be indicator functions for the behavioral tasks, i.e. set to 0 or 1 depending on the properties of each trial (Reference 1, Reference 8). For example, at time $k = 1$, $I_n(1) = [1 \ 0 \ 1 \ 1 \ 0]$.</p>
nIb	<p>A scalar that defines the dimension of the input to the discrete – binary – part of the model $y_d(k) = g_b(I_b(k), X(k), Q)$. $I_b(k)$ is the input vector with “nIb” elements, and the g function is defined by a sigmoid function:</p> $P(y_d(k) == 1) = \exp(ET_k X(k) + FT_k) / (1 + \exp(ET_k X(k) + FT_k))$ <p>Note that ET_k and FT_k are functions of $I_b(k)$ and Q. The format of ET_k and FT_k is defined by the other input arguments (Ib, $DLink$, xM) to the <code>compass_create_state_space</code> function. Check xM, $DLink$ and $DLinkUpdate$ for more explanation.</p> <p>Example: $nIb = 5$, for instance $I_d(k) = [I_b(k) \ I_c(k) \ I_h(k) \ I_{i2c}(k) \ I_{c2i}(k)]$ has dimension of 5. The “I_b, I_c, I_h, I_{i2c}, and I_{c2i}” are indicator functions for the behavioral task which are the input to the discrete part of the behavioral model. For example, at time $k = 1$, $I_d(1) = [1 \ 0 \ 1 \ 1 \ 0]$.</p>
xM	<p>A transform matrix which determines how the state variables are used in both $y_c(k)$ and $y_d(k)$ observation processes. The default transform matrix is an identity $nx \times nx$ matrix, but it can be a rectangular matrix with a larger number of rows than its columns. Elements of the xM are 0 and 1, and the sum of each row is equal to 1.</p> <p>Note that rather than using $X(k)$ in the continuous or discrete process, we use $xM \times X(k)$. Using the xM, we have more flexibility in defining the observation processes used in the model. When xM is an identity matrix, each state variable can be linked to one of the input elements, whereas using a rectangular xM, we will be able to define behavioral models for which each state variable can be linked to more than one input element. For example, we might have the following behavioral model:</p> $y_c(k) = (I_c(k) + \beta_0 * I_h(k)) * x(k) + I_h(k) + v$ <p>here, the state variable is linked to two input elements. To build this model, we need to set $xM = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$; thus, the state variable – $nx = 1$ here – is linked to both $I_c(\cdot)$ and $I_h(\cdot)$. Note that $I_c(\cdot)$ and $I_h(\cdot)$ will be columns of In input passed to the model.</p> <p>Default: an identity $nx \times nx$ matrix</p> <p>Example: $xM = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$, the state variable utilized in the behavioral model is treated as a three-component state variable, where the second and third components are the same. Note that the nx must be 2 here.</p>
cLink	<p>A matrix of size $1 \times tx$, where tx is the number of xM rows. In this toolbox, we assume that there are tx state variables rather than nx, and $cLink$ defines which of I_n's columns is linked to each state variable – check In in <code>compass_em</code> function. Elements of the matrix can have values between 1 to nIn. I_n is the input to the continuous observation process with size of $K \times nIn$, where K is the length of process or number of trials.</p> <p>Example: $cLink = [1 \ 3 \ 4]$, this means there are 3 state variables, and the first state variable is linked to the first column of I_n and so on. This implies that, nIn should be at least 4. The $cLink$ suggests that the dynamic part of the continuous process will be defined by:</p>

	$I_{n,1}(k)x_1(k)\beta_1 + I_{n,3}(k)x_2(k)\beta_2 + I_{n,4}(k)x_3(k)\beta_3$ <p>here, $(\beta_1, \beta_2, \beta_3)$ are model parameters, and $CT_k = [I_{n,1}(k) * \beta_1 \quad I_{n,3}(k) * \beta_2 \quad I_{n,4}(k) * \beta_3]$. We will explain in the cLinkUpdate definition, which defines which of the parameters are fixed or should be estimated using the learning step.</p>
cLinkUpdate	<p>A matrix of size $1 \times tx$, similar to cLink. Elements of the matrix are 0 and 1, and it determines whether the corresponding parameter - β_i - will be updated or not, during the compass_em. A value of 1 means β_i will be updated, and value of 0 means β_i is fixed. The default value for the fixed parameters is 1. Check the cLink argument description. Note that cLink defines the model structure and then using cLinkUpdate function we determine which of the model parameters needs to be estimated in the learning step.</p> <p>Example: <code>cLinkUpdate = [1 0 0]</code>, this means only the first variable will be updated using EM algorithm. Note that this refers to the parameter linked to the first state variable, not to the first column of $I_n(k)$. Check cLink example, this means the continuous part of the continuous process is defined by:</p> $I_{n,1}(k)x_1(k)\beta_1 + I_{n,3}(k)x_2(k) + I_{n,4}(k)x_3(k)$ <p>In the other words, it means $\beta_2 = \beta_3 = 1$ – fixed values, it can be any other values as well – and β_1 will be adjusted through the EM algorithm.</p>
dLink	<p>dLink is equivalent to cLink for discrete – or binary - process. Check the cLink for further information.</p> <p>Note that we could set these parameters with dummy or null values if there are no discrete processes. The setting won't be used when we the learning – EM – function later.</p>
dLinkUpdate	<p>dLinkUpdate is equivalent to cLinkUpdate for discrete process. Check the cLinkUpdate for further information.</p> <p>Note that we could set these parameters with dummy or null values if there are no discrete processes. The setting won't be used when we the learning – EM – function later.</p>

Output Arguments

Param	<p>A structure array that defines the state-space and behavioral model structure and paramaters. Note that most \ elements are filled with their default values and they need to be set either manually or by calling compass_em function.</p> <p>The Param structure fields are:</p> <ul style="list-style-type: none"> ▪ nx – Check the input argument ▪ cLinkMap – It is equal to cLink ▪ dLinkMap – It is equal to dLink ▪ nln – Check the input argument, number of ln elements. ▪ nlb – Check the input argument, number of lb elements.
-------	--

- **Ak** – The state-transition model is defined by:

$$X(k+1) = A X(k) + B U(k) + W$$

The Ak defines the *A* variable of the state-transition model.

- **Bk** – the state-transition model is defined by:

$$X(k+1) = A X(k) + B U(k) + W$$

The Bk defines the *B* variable of the state-transition model.

- **Wk** – the state-transition model is defined by:

$$X(k+1) = A X(k) + B U(k) + W$$

The W defines the noise covariance term, and it is assumed to be stationary with a zero-mean.

- **X0** – The state-variable estimate at time 0. It is assumed that state variable at time 0 has a Normal distribution with mean X0 and variance W0.
- **W0** – The covariance matrix of the state-variable estimate at time 0.
- **Ck** – Model parameters for the CT_k section of the continuous observation process:

$$y_c(k) = CT_k X(k) + DT_k + v$$

Note that CT_k at time *k* is defined by: $(C_k * In(k, cLink)) \times xM$ - * is an element-by-element multiplication, and \times is a dot product.

- **Dk** – Model parameters for the DT_k of the continuous observation process:

$$y_c(k) = CT_k X(k) + DT_k + v$$

Note that DT_k at time *k* is defined by: $(D_k * DMk) \times In(k, :)'$.

DMk elements are either 0 or 1, and it defines which columns of the *In* contribute in the stationary part of the observation process.

DMk is equal to zero for the columns identified in *cLink*; thus, the stationary part of the observation process is defined by a linear combination of input columns not being linked to state variables.

Note the *DMk* is an internal parameter of the toolbox, which is derived from *cConstantUpdate*.

Note that, if we want to have a term similar to: $I_c(k) * x(k) + \beta_0 * I_c(k)$ in the observation process, we should replicate $I_c(k)$ twice in the *In* definition.

- **cConstantUpdate** – It is a vector of length *nIn*, and it determines which elements of D_k will be set to zero. Elements of the *cConstantUpdate* are either 0 or 1, where elements with value 1 determines columns of *In* used in the stationary part of the observation process. Note that *DMk* is

	<p>derived using the cConstantUpdate. The cConstantUpdate is 1 for the input columns not linked to the state variables and it is set 0 for the input columns lined to the state variables.</p> <ul style="list-style-type: none"> ▪ Vk – The variance for the continuous observation noise, <i>v</i>. This will be dispersion term for the Gamma distribution. Check Reference 1 for further information. ▪ S – It defines a constant shift in the observation signal and it is used for Gamma distribution. Check Reference 1 for further information. ▪ Ek – It is equivalent to Ck, and it is used for the discrete part of the state-space model. ▪ Fk – It is equivalent to Ek, and it is used for the discrete of the state-space model. ▪ dConstantUpdate – It is equivalent to cConstantUpdate, and it is used for the discrete part of the model. ▪ xM – check the input argument
--	--

Example

Param = [compass_create_state_space](#)(3,1,4,4,eye(3,3),[1 3 4],[1 1 1],[1 3 4],[0 0 0]);

Order of the input parameters to create the state space model:

1. The state-variable dimension is set to be 3.

$$X(k) = [x_1(k) \ x_2(k) \ x_3(k)]'$$

2. The input to the state-space transition model is a scalar, length 1.
3. Given items in 1 and 2, the state transition process is:

$$X(k+1) = AX(k) + bU(k) + W$$

4. The input to both the continuous and discrete model of the behavior are vectors with 4 columns and K rows, where K is the number of trials
5. The **xM** is an identity matrix, thus we have a state vector with 3 elements.
6. The [cLink](#) is [1 3 4], which means the first state variable is linked to the 1st element of I_n , the second state variable is linked to 3rd element of I_n , and the third state variable is linked to the 4th element of I_n .
7. The [cLinkUpdate](#) is [1 1 1], which means all three *C* parameters will be updated. Check [cLinkUpdate](#).
8. Given items 6 and 7, the observation process for a normal distribution is defined by:

$$y_c(k) = c_1 \ln(k, 1)x_1(k) + c_2 \ln(k, 3)x_2(k) + c_3 \ln(k, 4)x_3(k) + d_1 \ln(k, 2) + w$$

9. The [dLink](#) is [1 3 4], which share a similar structure of the continuous model of the behavior.
10. The [dLinkUpdate](#) is [0 0 0], which means none of the *E* parameters will be updated. Check [dLinkUpdate](#).
11. Item 10 is very important to avoid mis-specified models of the observed signals.

12. Given items 9 and 10, the observation process for the discrete process is defined by:

$$\text{Logit}(P(y_b(k) == 1)) = lb(k, 1)x_1(k) + lb(k, 3)x_2(k) + lb(k, 4)x_3(k) + f_1 \ln(k, 2)$$

Note, E_k elements are assumed to be equal to 1.

13. Check items 8 and 11 to find the relationship between [cLink](#), [cLinkUpdate](#), [dLink](#), and [dLinkUpdate](#) with the process model. Note that [cLinkUpdate](#) and [dLinkUpdate](#) must be carefully chosen, otherwise we might have a non-identifiable model.

Note

The `compass_create_state_space` sets the `Ak`, `Bk`, `Wk`, `W0`, `Ck`, `Dk`, `Vk`, `S`, `Ek` and `Fk` parameters with predefined values. Values of these parameters can be changed either inside this function or after calling this function; the other output arguments mainly define the model structure and it shouldn't be changed manually.

We will see later that some of these parameters will be updated, while some are fixed. These parameters can be changed inside the `compass_create_state_space` function or even manually. Specifically, the `cConstantUpdate` and `dConstantUpdate` are defined by a specific assumption of how the input to state-space model will be linked to constant parameters or state variables, which is part of the `compass_create_state_space` function. We can change the structure of the behavioral model manually by changing elements of `Param` variables.

compass_set_learning_param

It sets learning parameters used in the EM algorithm, i.e. which parameters are considered fixed vs. updated based on data. Note this function can be only called after [compass_create_state_space](#) function. Using this function, we determine how the parameters of the models will be updated later using [compass_em](#) function. Note that [compass_create_state_space](#) defines a proper structure of the model, while the [compass_set_learning_param](#) sets how parameters of the state-space model will be trained using the Expectation-Maximization (EM) algorithm.

Syntax

```
Param = compass_set_learning_param(Param,Iter,UpdateStateParam,UpdateStateNoise,UpdateStateX0,UpdateCModelParam,  
UpdateCModelNoise, UpdateDModelParam,DiagonalA,UpdateMode,UpdateCModelShift)
```

Description

```
Param = compass_set_learning_param(Param,Iter,UpdateStateParam,UpdateStateNoise,UpdateStateX0,UpdateCModelParam,  
UpdateCModelNoise, UpdateDModelParam,DiagonalA,UpdateMode,UpdateCModelShift)
```

This function sets how the model parameters will be adjusted using the EM learning procedure run by [compass_em](#).

Input Arguments

Param	<p>A structure array consisting both the structure and parameters of the state-space model parameters along with EM learning procedure.</p> <p>Check Param in compass_create_state_space for a more complete description.</p>
Iter	<p>Number of EM training iterations.</p> <p>Note that the value of <i>Iter</i> can be determined by inspecting the ML curve produced by compass_em. After visualizing the ML curve, we can reset the <i>Iter</i> to make sure that ML converges to a local maximum.</p> <p>Example: <i>Iter</i> = 100. The EM training iteration number is 100.</p>
UpdateStateParam	<p>It can be either 0 or 1.</p> <p>A value of 1 means that the state-transition model parameters, elements of A and B matrices, will be updated. A value of 0 means that these parameters are not updated in the EM training process.</p> <p>State transition process model is defined by:</p> $X(k+1) = \textcolor{red}{A} X(k) + \textcolor{red}{B} U(k) + W$ <p>Example: <i>UpdateStateParam</i> = 1. The state-transition process A and B matrixes will be updated in the EM learning process.</p> <p>Note that we either update both <i>A</i> and <i>B</i> or neither.</p>

	Training only A or B would require a change in the compass_em code.
UpdateStateNoise	<p>It can be either 0 or 1.</p> <p>Value of 1 means that the state-transition model covariance matrix will be updated. It is assumed that W, a covariance matrix, is diagonal. Value 0 means that W won't be updated in the training process.</p> <p>State transition process model is defined by:</p> $X(k+1) = A * X(k) + B * U(k) + W$ <p>Example: <i>UpdateStateNoise</i> = 1. The state-transition process parameters are trained in the EM learning process.</p>
UpdateStateX0	<p>It can be either 0 or 1.</p> <p>Value of 1 means that the state variable parameters at time 0 – mean and covariance matrix - will be updated. Value 0 means that the initial parameters of the state variable at time 0 won't be updated in the EM training process. The initial value of X_0 are set inside compass_create_state_space. If <i>UpdateStateX0</i> is set to 0, the user should manually initialize the state by accessing <code>Param.X0</code> and <code>Param.W0</code>.</p> <p>State transition process model is defined by:</p> $X(k+1) = A * X(k) + B * U(k) + W$ $X(0) \sim N(X_0, W_0)$ <p>Example: <i>UpdateStateX0</i> = 1. It means X_0 and W_0 – mean and covariance matrix - of the initial value of the state-variable is estimated in compass_em function.</p>
UpdateCModelParam	<p>It can be either 0 or 1.</p> <p>Value of 1 means parameters of the continuous observation process will be updated in the EM training process. The parameters are a sub-set of parameters in the C set to be trained and all the parameters in D vectors. Read the cLinkUpdate in compass_create_state_space function to find which subset – or index - of the C parameters will be updated.</p> <p>Value of 0 means parameters of the continuous observation process won't be updated in the EM training process.</p> <p>Example: <i>UpdateCModelParam</i> = 1. It means parameters of the continuous observation process – elements of C and D vectors – will be updated in the EM training process.</p>
UpdateCModelNoise	<p>It can be 0 or 1.</p> <p>Value of 1 means the observation process noise-term will be updated through the EM training process, compass_em. For the continuous observation process with a normal or log-normal distribution, the noise term – v – represents additive Normal white</p>

	<p>noise. For the observation process with a Gamma distributed signal, the v term corresponds to the dispersion term. For further information about the canonical form of the Gamma distribution and dispersion term definition, please see Reference 1.</p> <p>For the Normal observation process, we have:</p> $y_c(k) = CT_kX(k) + DT_k + v$ $v \sim N(0, \sigma_v^2)$ <p>here, v is a white noise with a Normal distribution with a variance of σ_v^2.</p> <p>Example: <i>UpdateCModelNoise</i> = 1. It means the noise-term of the observation process is updated through the EM training process.</p>
UpdatedModelParam	<p>It can be 0 or 1.</p> <p>It is equivalent to the UpdateCModelParam , but applies to a discrete observation process. Check the UpdateDModelParam for further explanations.</p> <p>Note that for the discrete process, there is no noise-term to be updated.</p>
DiagonalA	<p>It can be 0 or 1. Default is 1.</p> <p>Value 1 means the A matrix in the state-transition process model will be a diagonal matrix. The diagonal assumption is the favorable choice for EM learning. Note that state variables will be dependent on each other through the observation process even if A is a diagonal. Also note that the posterior estimates of the state variables are not necessarily independent, because they are updated through the observation model.</p> <p>Value 0 means the A matrix is a full matrix. For the EM algorithm, this might lead to a less accurate or even instable estimation of both parameters and state-variables.</p> <p>Note that whether A is updated from data during the EM process will be determined by UpdateStateParam. The state transition process can be either an AR(1) model or a random walk. For many applications, we will want to set the state-transition process as a random-walk type process, where we set A to be an identity matrix and exclude it from the training process. This does not mean that the state is fixed – the state will still change trial-to-trial based on the posterior updates (which in turn depend on the observations). It means that the state only changes based on the observations, not based on its own internal dynamics. Random-walk state processes are useful whenever we are interested in understanding how a cognitive process evolves through time.</p> <p>We will have an AR(1) state model whenever A is something other than an identity matrix. This models a state that naturally settles over time to a default value. This might be appropriate if the state should naturally decay to 0 over time, such as a forgetting process or the decaying effect of something like a reward/punishment. It will often be helpful to then define an input term to the state process, and this input term will be a major driver of state change.</p> <p>Example: <i>DiagonalA</i> = 1 – default value. It means the A matrix is a diagonal matrix.</p>
UpdateMode	<p>It can be 1 or 2. Default is 2.</p>

	<p>It determines which update method is used in the Filter mean and covariance update. Note that when we use a Gaussian approximation method, we generally might have different update rules for the mean and co-variance of the state variables. We are using two different update rules:</p> <ol style="list-style-type: none"> 1) First update Mean, and then update Covariance - UpdateMode 1 2) First update Covariance, and then update Mean – UpdateMode 2 <p>Picking the right update rule depends on the observation process and its characteristics, specifically its variability and noise. In practice, the good starting point is UpdateMode 2. To identify the better update rule, we can check both the estimation and ML result using both update rules, then select the one which gives a higher ML or a more stable estimation result.</p> <p>For a further detail about which of these approximations might give a better result, you can check Reference 2. Reference 2 describes the Gaussian approximation for the point process model, which can be extended for other distributions including Gamma, Bernouli, and or mixture of these distributions.</p> <p>Example: <i>UpdateMode</i> = 2 – default value. It means the Filter estimate of the state-variable is based on the second method used in the posterior Gaussian approximation.</p>
UpdateCModelShift	<p>It can be 0 or 1.</p> <p>It determines whether the estimation of a positive shift – or bias - will be part of the EM training process or not. For the Gamma distribution, we define a positive shift and we can tune this parameter using the EM algorithm. A value of 1 means that the shift parameter will be updated in the EM training process.</p> <p>A value of 0 means that this shift parameter is pre-set and it is not updated in the EM training process, compass_em function.</p> <p>Note that this is a specific parameter of the Gamma distribution, and it is not used in the observation process with Normal or Log-Normal distributions.</p> <p>For a further information about the Gamma distribution, please check Reference 1.</p> <p>Example: <i>pdateCModelShift</i> = 1 . The time shift parameter will be updated if the continuous observation process is set to be Gamma.</p>

Output Arguments

Param	<p>An updated structure array consisting of both the state-space model structure/parameters and EM learning parameters.</p> <p>The new fields of the Param structure are:</p> <ul style="list-style-type: none"> ▪ Iter – check the input argument ▪ UpdateStateParam – check the input argument ▪ UpdateStateNoise – check the input argument ▪ UpdateStateX0 – check the input argument
-------	---

	<ul style="list-style-type: none"> ▪ UpdateCModelParam – check the input argument ▪ UpdateCModelNoise – check the input argument ▪ UpdateDModelParam – check the input argument ▪ DigonalA – check the input argument ▪ UpdateMode - check the input argument ▪ UpdateCModelShift - check the input argument <p>Note that Param will also include the state-space model parameters defined by compass_create_state_space as well.</p>
--	---

Example

```
% create state space model
Param = compass_create_state_space(3,1,4,4,eye(3,3),[1 3 4],[1 1 1],[1 3 4],[0 0 0]);

% set the learning parameters
Iter = 300;
Param = compass_set_learning_param(Param,Iter,0,1,1,1,1,1,2,1);
```

In this example, we set Number of EM training iterations to 300. Given the input terms to the [compass_set_learning_param](#), the state parameters are fixed and not updated in the EM step. Similarly, we can define whether other elements of the model will be updated or not by corresponding values being passed to [compass_set_learning_param](#).

Note

In [compass_create_state_space](#), we determine which part of the proposed dynamical model will have free parameters. Using the [compass_set_learning_param](#) function, we determine if these free parameters will be updated or not.

For the state-transition model, we can:

1. Set whether A will be diagonal or full matrix
2. Set whether elements of A and B matrices will be updated or not – Both A and B will determine the state-transition dynamics, and for this reason, we either update both A and B or none of them. Separate updating of only A or B will be available in a future version of COMPASS.
3. Set whether the noise covariance matrix – which is assumed to be diagonal – will be updated or not.
4. Set whether X0 variable distribution will be updated or not.

For the continuous behavioral model, we can:

1. Set whether model parameters will be updated or not. This is a selected subset of the C and all D elements.
2. Set whether observation noise will be updated or not.
3. For the Gamma distribution, we can determine whether the shift term will be updated or not.

For the discrete behavioral model, we can set whether model parameters will be updated or not.

compass_set_censor_threshold_proc_mode

It sets censor threshold for the continuous observation process and how it will be processed in the training step. The censoring threshold will be added to the `Param` structure. We assume that on the censored trials, the continuous observation process exceeds the censoring threshold and that both the continuous and discrete observations are unobserved. Check `obs_valid` argument in the `compass_em` for further definition of the censored or missed trials.

Syntax

```
Param = compass_set_censor_threshold_proc_mode(Param,censor_thr, censor_mode,update_mode)
```

Description

```
Param = compass_set_censor_threshold_proc_mode (Param,censor_thr,censor_mode,update_model)
```

This function gets the `Param` structure plus the censored value, `censor_thr`, and it updates the `Param` structure. The `censor_time` in the `Param` keeps the value of censoring threshold. The `censor_mode` and `update_model` define how the censored points will be processed in the training step. At present, imputation of censored points is only available for a univariate continuous observation; vector-valued continuous observations cannot be imputed through this function.

Input Arguments

Param	<p>A structure array consisting both the state-space and observation process model parameters along EM learning parameters.</p> <p>Check <code>Param</code> in <code>compass_create_state_space</code> and <code>compass_set_learning_param</code> for a more complete description.</p>
censor_thr	<p>It define the continuous observation process censor time.</p> <p>The <code>censor_time</code> variable of the <code>Param</code> will be equal to <code>censor_thr</code>.</p> <p>Example: <code>censor_thr = 1</code>. It means that trials are censored whenever they have a continuous observation process value larger than 1.</p>
censor_mode	<p>It defines which processing mode is being for censored data points. There are two modes, 1 and 2.</p> <p>The <code>censor_mode</code> equal to 1 uses imputation (single imputation) to process censored data point. In this mode, the <code>compass_em</code> process will treat the imputed points as though they were actually observed. In other words, missing observations will not affect the confidence bounds of the state estimate.</p> <p>The <code>censor_mode</code> equal to 2 uses a Gaussian approximation, to approximate the full likelihood of censored data. This is the method described in Reference 3. As in that paper, the state estimate will have wider confidence bounds on missing data, because the uncertainty is propagated through.</p>

	<p>Example: <code>sensor_mode = 2</code>. It means the censored trials are processed and the process values imputed using a full-likelihood method.</p>
<code>update_mode</code>	<p>It defines how the fast Gaussian solution will be run on the censored data points to impute them under <code>sensor_mode = 2</code>. There are two modes, 1 and 2</p> <p>This setting is similar to <code>UpdateMode</code> defined for <code>compass_set_learning_param</code> and defines how the filter solution is called over the censored data points.</p> <p>When <code>sensor_mode</code> is set to 1, we impute data for the censored trials and the data point is treated as an observed point. Thus, this setting is only effective when we use the Gaussian approximation mode (<code>sensor_mode = 2</code>) for processing censored data points.</p> <p>Example: <code>update_mode = 1</code>. It means that missing censored data will be imputed using a Gaussian filtering approximation, where first the mean and then the covariance is updated during the EM process.</p>

Output Arguments

Param	<p>An updated structure array of <code>Param</code> consisting of model structure, model parameters and EM learning parameters. The new field of the <code>Param</code> structure is: <code>sensor_time</code> - check the input argument</p> <p>The following arguments are being added to the <code>Param</code>.</p> <p><code>Param.sensor_time</code>: it keeps the censoring threshold <code>Param.sensor_mode</code>: it keeps the <code>sensor_mode</code> argument <code>Param.sensor_update_mode</code>: it keeps <code>update_mode</code> argument</p>
-------	--

Example

```

sensor_time = 1;
sensor_mode= 2;
update_mode= 1;
Param= compass_set_censor_threshold_proc_mode (Param,sensor_time, sensor_mode,
update_mode);

```

This means that the censored trials will have values larger than `sensor_time` which is 1 here. The `full likelihood methodology` will be called on the censored data points, and the `update_mode` is type 1.

compass_em

This function runs the EM algorithm to adjust model parameters and estimate the state variables. This is the **core** function of the toolbox. It takes the model structure, input and observation data and it returns updated model parameters and state estimates. It also returns the Maximum Likelihood (ML) estimate which shows progression of the ML through multiple iterations.

Syntax

```
[rXSmt,rSSmt,Param,rXPos,rSPos,ML,EYn,EYb,rYn,rYb]=compass_em(DISTR,Uk,In,Ib,Yn,Yb,Param,obs_valid)
```

Description

```
[rXSmt,rSSmt,Param,rXPos,rSPos,ML,EYn,EYb,rYn,rYb]= compass_em(DISTR,Uk,In,Ib,Yn,Yb,Param,obs_valid)
```

The function gets input arrays (**Uk,In,Ib**), observation signals (**Yn,Yb, obs_valid**), model parameters (**Param**), and type of observation processes (**DISTR**), and returns an estimate of state variables (**smoother** - **rXSmt,rSSmt** -, plus **filter** - **rXPos,rSPos**) along with the likelihood value on each iteration (**ML**), expected observed signals (**EYn,EYb**), and updated observed signals along with imputed data for censored trials (**rYn,rYb**).

The **filter estimate** uses past and current observations to predict the current state. It is causal – the state at each timepoint is only influenced by prior observations.

The **smoother estimate** uses all the observations – past and future - to predict the current state. Smoother output is non-causal, but may give a more accurate estimate of the underlying process and is usually more suitable for off-line analysis.

Input Arguments

DISTR	<p>A 2 element vector which defines the type of the observed signals and their corresponding distributions.</p> <p>DISTR=[1 0] means there is only continuous observation signal and the proper distribution for the signal is Normal. Note that the input argument Yn carries the continuous signal.</p> <p>DISTR=[2 0] means there is only continuous observation signal and the proper distribution of the signal is Gamma. Note that the input argument Yn carries the continuous signal.</p> <p>DISTR=[0 1] means there is only discrete observation – a Bernoulli observation process. Note that the input argument Yb carries the continuous signal.</p> <p>DISTR=[1 1] means there are both continuous and discrete observation signals and the proper distribution for the signal is a mixture of Normal and Bernoulli distributions. Note that the input arguments Yn and Yb carry the mixed signals.</p> <p>DISTR=[2 1] means there are both continuous and discrete observation signals and the proper distribution for the signal is a mixture of Gamma and Bernoulli distributions. Note that the input arguments Yn and Yb carry the mixed signals.</p>
-------	---

	<p>Example: $DISTR = [1 \ 0]$, it means the observed signal includes only continuous observation. Note that we might define the state-space model with both continuous and discrete observations; the EM algorithm might process either continuous or discrete parts of the process given the $DISTR$ elements.</p>
Uk	<p>A matrix of size $K \times nUk$ which defines the input to the state-transition model – K is the length of observation, and nUk is number of elements per each time index. The state-transition process is defined by:</p> $X(k+1) = A X(k) + B U(k) + W$ <p>Example: $Uk = ones(100,1)$, it mean that input is a fixed value of 1 at each time index. Note that it is assumed that K is equal to 100, which is the length of the observation.</p>
In	<p>A matrix of size $K \times nIn$ which defines the input to the continuous observation process. For further information about how columns of In is utilized in the continuous observation process, check the compass_create_state_space function.</p> <p>Example: $In = randn(100,5)$; it means a random input with 5 elements at each time index is passed to the continuous observation process. Note that it is assumed that K is equal to 100, which is the length of the observation.</p> <p>Practically, the In elements could be indicator functions of the task factors or even continuous values, including history terms.</p> <p>Note that Uk, In, and Ib share the same number of rows.</p>
Ib	<p>A matrix of size $K \times nIb$ which defines the input to the discrete observation process. It is equivalent to In of the continuous part.</p> <p>Example: $Ib = randn(100,5)$; it means a random input with 5 elements at each time index is passed to the discrete observation process. Note that it is assumed that K is equal to 100, which is the length of the observation.</p> <p>Practically, the Ib elements could be indicator functions of the task factors or even continuous values, including history terms.</p> <p>Note that Uk, In, and Ib share the same number of rows.</p>
Yn	<p>A vector of size $K \times 1$ which carries the continuous observation signal - K is the length of observation process.</p> <p>Values of Yn can real values or NaN. Note that for a NaN observation signal, the corresponding obs_valid must be either 0 or 2. Check obs_valid for further information.</p> <p>Note that for an observation process with a Normal distribution, $DISTR = [1 \ X]$, the Yn elements may be any real values. For the observation process with a Gamma distribution, $DISTR = [2 \ X]$, the Yn elements must be positive real-valued.</p> <p>Example: $Yn = rand(100,1)$; it means a positive random input with 100 rows.</p>

	Note that U_k , I_n , I_b , and Y_n share the same number of rows.
Yb	<p>A vector of size $K \times 1$ which carries the discrete observation signal - K is the length of observation process.</p> <p>Values of Yb are 0, 1, or NaN. Note that for a NaN observation signal, the corresponding obs_valid must be either 0 or 2. Check obs_valid for further information.</p> <p>Value of 0 means an incorrect response, and a value of 1 means a correct response.</p> <p>Example: $Yb = randi(2, [100\ 1]) - 1$, which means a vector with random values of 0 or 1 at each row.</p> <p>Note that U_k, I_n, I_b, Y_n, and Yb share the same number of rows.</p>
Param	<p>A structure array consisting of both the state-space model structure plus EM learning parameters.</p> <p>Check Param in compass_create_state_space and compass_set_learning_param for a more complete description.</p>
obs_valid	<p>A vector of length K with elements of 0, 1, and 2.</p> <p>An element with value 1 means the observed signal is valid. An element with value 0 means the observation is missing. An element with value 2 means the observation is censored. The censoring threshold is defined by the Param.censor_time.</p> <p>Note that a NaN value in either Y_n or Y_b must be accompanied by a value 0 or 2 in obs_valid, otherwise the EM algorithm fails to have a correct estimate of the model parameters and state variables.</p> <p>Note that obs_valid determines how the EM algorithm treats each row of the observed signal in running either E- or M-step of the EM process. Thus, setting correct values of obs_valid has a significant importance in running the model correctly.</p> <p>Example: $obs_valid = ones(100,1)$, it means all elements of the observed signals are filled with valid values.</p> <p>Note that U_k, I_n, I_b, Y_n, Y_b, and obs_valid share the same number of rows.</p>

Output Arguments

rXSmt	<p>The mean value of the state variable smoother result.</p> <p>A structure array of length K, where K is the number of observation samples. Each element of the rXSmt is a vector of length nx.</p>
rSSmt	The covariance matrix of the state variable smoothing result.

	<p>A structure array of length K, where the K is number of observation samples. Each element of the <i>rSSmt</i> is a matrix of size $nx \times nx$.</p>
Param	<p>Updated <i>Param</i> structure array which returns EM learning result.</p> <p>Note that <i>Param</i> keeps all the model parameters.</p>
rXPos	<p>The mean value of the state variable filter result.</p> <p>A structure array of length K, where K is the number of observation samples. Each element of the <i>rXPos</i> is a vector of length nx.</p>
rSPos	<p>The covariance matrix of the state variable filter result.</p> <p>A structure array of length K, where the K is the number of observation samples. Each element of the <i>rSPos</i> is a matrix of size $nx \times nx$.</p>
ML	<p>Maximum likelihood value of the EM algorithm at each iteration.</p> <p>The ML is a vector of the length <i>Iter</i>. <i>Iter</i> is the number of EM iterations, and it is passed through <i>Param</i> to <i>compass_em</i>.</p> <p>Practically, the ML might be an increasing function of the iteration. In the other words, it is expected to grow as the number of iteration increases.</p> <p>ML growth rate or reaching a plateau might be an indication of enough number of EM iteration. The ML value can be used as the stop criterion of the algorithm; a future version of <i>compass_em</i> will contain this non-deterministic stopping . Note that the optimal number of <i>Iter</i> which is passed to the <i>compass_set_learning_param</i> can be defined by observing the ML curve growth over iterations.</p> <p>Because we start with a random start value for model parameters, there is a chance to see a non-increasing ML on early iterations of the algorithm. One approach would be to estimate a static version of the model, e.g. through a standard GLM with the same terms as the state-space model. The arguments of <i>Param</i> can then be manually set as described above. Note that changes in <i>Param</i> must be done after calling <i>compass_create_state_space</i> function. Using a reasonable starting point, we might reduce number of EM iterations. Furthermore, we might get a more well-behaved ML curve as well.</p>
Eyn	<p>A vector of length K which returns the expected value of the continuous process at each time index.</p> <p>In the Normal observation process, this is the deterministic part of the observation process estimated by the model – or simply the mean estimate of the process at each time index. This is defined by:</p> $y_c(k) = CT_k X_{k K}(k) + DT_k$ <p>In the Gamma observation process, this is equal to the mean of the observation process at each sample. Check Reference 1 for further information. This is defined by:</p> $y_c(k) = \exp(CT_k X_{k K}(k) + DT_k)$

	Note that parameters in CT_k and DT_k are updated parameters.
Eyb	<p>A vector of length K which returns the expected probability of the correct choice at each time index.</p> <p>The expected probability at each time index is defined by:</p> $P(y_d(k) == 1) = \exp(ET_k X_{k K}(k) + FT_k) / (1 + \exp(ET_k X_{k K}(k) + FT_k))$ <p>Note that parameters in ET_k and FT_k are updated parameters.</p>
rYn	<p>Updated Yn Vector. It has the same length as Yn.</p> <p>Elements of the vector corresponding to censored data are replaced by an imputation technique. Check compass_sampling for further information.</p>
rYb	<p>Updated Yb Vector. It has the same length as Yb.</p> <p>Elements of the vector corresponding to censored data are replaced by an imputation technique. Check compass_sampling for further information</p>

Example

Here, we assume input elements are properly loaded and assigned.

```
%% Load sample data
% data: li, Ini, lin, Yk_1, Yk
load('MSIT_1.mat');

%% First step, build state space model - create behavioral model
Param = compass_create_state_space(1,0,5,0,[1;1],[1 2],[0 1],[],[]);

%% Second step, define learning parameters
Iter = 100;
Param = compass_set_learning_param(Param,Iter,0,1,1,1,1,1,2,1);

%% Third step, define censored time threshold if it is necessary
censor_time = 1;
Param = compass_set_censor_threshold_proc_mode (Param,censor_time);

%% Fourth step, format the data (all points are observed)
% In
In = [Ini lin ones(length(Yk),1) li Yk_1];
% all data points are valid
valid = ones(length(Yk),1);

%% fifth Step, EM Algorithm
[XSmt,SSmt,Param,XPos,SPos,ML,Yp]=compass_em([2 0],[],In,[],Yk,[],Param,valid);
```

compass_filtering

This function runs the filtering (causal state estimation) algorithm, given the most recent observed signals. This function is appropriate for real-time filtering, when the observed signals are updated trial-by-trial. This function comes in handy when we want to run the toolbox in real time to assess changes in underlying state variable. The function gets the current observation signals plus an estimate of the state variable from the previous trial to estimate the current value of the state variables. The `Param` structure normally comes from `compass_em`, which should be run on a training dataset. That is, the best way to use this function is to have the subject perform a block of behavioral trials, run `compass_em`, and then use the results of that analysis to perform real-time behavior analysis.

The present version of `compass_filtering` can only run on scalar values of the continuous/discrete process, i.e., a single value such as reaction time and a single value such as correct/incorrect choice.

Syntax

```
[XPos,SPos,YP,YB]=compass_filtering(DISTR,Uk,In,Ib,Yn,Yb,Param,obs_valid,XPos0,SPos0)
```

Description

```
[XPos,SPos,YP,YB]=compass_filtering(DISTR,Uk,In,Ib,Yn,Yb,Param,obs_valid,XPos0,SPos0)
```

It runs an on-line filter algorithm on each new trial – or time-index – of the data.

Input Arguments

DISTR	Check <code>compass_em</code> for further information
Uk	<p>A vector of size $1 \times nUk$ which defines the current values of the input to the state-transition model.</p> <p>Note that <code>compass_filtering</code> will be called per trial, i.e. only one trial worth of input should be provided.</p> <p>For further information, check <code>compass_em</code> function</p> <p>Example: $Uk = 1$, it means the current input to the state-transition process is a scalar value of 1.</p>
In	<p>A vector of size $1 \times nIn$ which defines the current value of the input to the continuous process of the model.</p> <p>Note that <code>compass_filtering</code> will be called per trial.</p> <p>For further information, check <code>compass_em</code> function</p> <p>Example: $In = rand(1,5)$, it means a random input with 5 elements at each trial.</p>
Ib	A vector of size $1 \times nIb$ which defines current value of the input to the discrete process of the model.

	<p>Note that <code>compass_filtering</code> will be called per trial.</p> <p>For further information, check <code>compass_em</code> function</p> <p>Example: $Ib = rand(1,5)$, it means a random input with 5 elements on the current trial.</p>
Yn	<p>A scalar value; either a real value or NaN.</p> <p>For further information, check <code>compass_em</code> function definition.</p> <p>Example: $Yn = rand$, it means a random input for current observation of the continuous signal.</p> <p>Note that in practice, this is the observed behavior or an equivalent signal.</p>
Yb	<p>A scalar value either 0, 1, or NaN.</p> <p>For further information, check <code>compass_em</code> function definition.</p> <p>Example: $Yb = 1$, it means a correct response for the current trial of the discrete signal.</p> <p>Note that in practice, this is the observed behavior or an equivalent signal.</p>
Param	<p>Updated <code>Param</code> structure. In practice, this is the output of the <code>compass_em</code> run over a training dataset.</p> <p>A structure array consisting both the state-space and behavioral model parameters plus EM learning setup.</p> <p>Check <code>Param</code> in <code>compass_create_state_space</code> and <code>compass_set_learning_param</code> for a more complete description.</p>
XPos0	<p>The mean value of the state variable filter on the previous trial.</p> <p>The <code>XPos0</code> is a vector of length nx.</p>
SPos0	<p>The covariance matrix of the state variable filter on the previous trial.</p> <p>The <code>sPos0</code> is a matrix of size $nx \times nx$.</p>
obs_valid	<p>A scalar variable with values of either 0, 1, or 2.</p> <p>A value of 1 means a valid observed signal/s. A value of 2 means a censored observed signal/s. A value of 0 means a missing observed signal/s.</p> <p>For further information, check the <code>compass_em</code> function definition.</p>

Output Arguments

Xpos	<p>The mean value of the state variable filter on the current trial.</p> <p>Note that it is a function of the previous trial state estimate plus the new observed signals.</p> <p>The <i>XPos</i> is a vector of length nx.</p>
Spos	<p>The covariance matrix of the state variable filter on the current trial.</p> <p>Note that, it is a function of the previous trial state estimate plus the new observed signals.</p> <p>The <i>SPos</i> is a matrix of size $nx \times nx$.</p>
YP	<p>A scalar value which returns the expected value of the continuous process on the current trial.</p> <p>It is a function of the current estimate of the state variable and the trial input variables – Uk and In.</p>
YB	<p>A scalar value which returns the expected probability of correct decision on the current trial.</p> <p>It is a function of the current estimate of the state variable and the trial input variables – Uk and Ib.</p>

Example

Here, we assume the *Param* is the output of the `compass_em` algorithm ran over the training dataset. `compass_filtering` is called twice, once for the first trial and then on the second trial of the task. Note how we set our initial estimates of the mean and covariance of the state variables. If we had training data, we could equally set these based on the end of that training data. We can put the following procedure in a loop, where it would be called on each subsequent trial.

```
% Set initial state and variance; seed values are required for the algorithm
XPos0=[0;0];
SPos0=[10 0;0 10];

% first observation (note In,Ib are vectors of length 1xnIn, 1xnIb and similarly Uk)
% XPos0 and SPos0 are the estimate of state mean and variance from previous one
% we assume data is valid
[XPos0,SPos0]=compass_filtering([1 0],[],In(1,:),[],Yn(1,[],Param,1,XPos0,SPos0);

% second observation
[XPos0,SPos0]=compass_filtering([1 0],[],In(2,:),[],Yn(2,[],Param,1,XPos0,SPos0);
```

compass_sampling

This function is called to generate a sample of continuous and discrete signals on the censored trials. This function is used in the imputation technique used in [compass_em](#) on the censored trials. The current version of [compass_em](#) fetches a sample of data given the current update of the observation distribution on the censored trials, then runs the EM algorithm using the imputed sample.

Syntax

```
[Yn,Yb]=compass_sampling(DISTR,Cut_Time,Uk,ln,lb,Param,XPos0,SPos0)
```

Description

```
[Yn,Yb]=compass_sampling(DISTR,Cut_Time,Uk,ln,lb,Param,XPos0,SPos0)
```

It generates a sample ([Yn,Yb](#)) for continuous and discrete observation signals, given the current inputs of the model ([Uk,ln,lb](#)) and a previous estimate of the state-variables ([XPos0,SPos0](#)). The [Cut_Time](#) variable defines the censoring threshold; note that censoring criteria is only conditioned on the value of the continuous signal. [Param](#) keeps the updated set of the model parameters.

Input Arguments

DISTR	It is similar to DISTR defined in compass_filtering and compass_em . Check compass_em or compass_filtering for further information
Cut_Time	It is a scalar value which define the censoring threshold. A copy of this value is carried by Param as well.
Uk	It is similar to Uk defined in compass_filtering . Check compass_filtering for further information.
ln	It is similar to ln defined in compass_filtering . Check compass_filtering for further information
lb	It is similar to lb defined in compass_filtering . Check compass_filtering for further information .
Param	It is similar to Param defined in compass_filtering . Check compass_filtering for further information
XPos0	It is similar to XPos0 defined in compass_filtering . Check compass_filtering for further information
SPos0	It is similar to SPos0 defined in compass_filtering . Check compass_filtering for further information

Output Arguments

Yn	A scalar variable returning a sample of the continuous signal over the censored threshold.
Yb	A scalar discrete variable returning a sample of the discrete signal over the censored threshold. Yb can be either 0 or 1.

Example

Here, the `compass_sampling` is called inside the `compass_em` function to create sample data to impute the state for the censored trials.

```
if obs_valid(k)==2
    [tYP,tYB]=compass_sampling(DISTR,censor_time,Uk(k,:),ln(k,:),lb(k,:),Param,XPre{k},SPre{k});
    if DISTR(1)>0    Yn(k)=tYP; end;
    if DISTR(2)==1  Yb(k)=tYB; end;
end
```

compass_deviance

This function can be called to estimate the deviance measure over both continuous and discrete parts of the observations. Deviance is an extension of the concept of mean squared error, which is used in the linear models with a normal observation process. This function requires [compass_em](#)'s output argument and it can be called after that. The deviance measure can be used to compare goodness-of-fit across different models, and to identify the model which gives a better fit.

Syntax

```
[DEV_C, DEV_D]=compass_deviance(DISTR, In, Ib, Yn, Yb, Param, obs_valid, XSmt, SSmt)
```

Description

```
[DEV_C, DEV_D]=compass_deviance(DISTR, In, Ib, Yn, Yb, Param, obs_valid, XSmt, SSmt)
```

It generates deviance measures ([DEV_C](#), [DEV_D](#)) for continuous and discrete observation signals, given the smoother estimate of the state variables plus other inputs to the model ([In](#), [Ib](#)). [Param](#) keeps the updated set of the model parameters. [obs_valid](#) carries information of each data point: whether it is observed, censored or dropped (missing at random). Note that deviance calls [compass_sampling](#) on censored data points and ignores missing at random data points in calculating deviance. We do not require [Uk](#) as an input argument. [Uk](#) determines the state transition process, and we have the state estimation through [XSmt](#) and [SSmt](#). Thus there is no need for [Uk](#).

Input Arguments

DISTR	It is similar to DISTR defined in compass_filtering and compass_em . Check compass_em or compass_filtering for further information
Cut_Time	It is a scalar value which define the censoring threshold. A copy of this value is carried by Param as well.
In	It is similar to In defined in compass_filtering . Check compass_filtering for further information
Ib	It is similar to Ib defined in compass_filtering . Check compass_filtering for further information .
Param	It is similar to Param defined in compass_filtering . Check compass_filtering for further information
XSmt	It is similar to XSmt defined in compass_em output argument. Check compass_em for further information
SSmt	It is similar to SSmt defined in compass_em output argument. Check compass_em for further information

Output Arguments

DEV_C	A scalar variable returning the deviance measure for the continuous observation process. This is a valid value only when DISTR is set for continuous observation.
DEV_CD	A scalar variable returning the deviance measure for discrete observation process. This is a valid value only when DISTR is set for discrete observation.

Example

Here, **compass_deviance** is called after **compass_em** to estimate the deviance measures for both discrete and continuous parts.

```
%% Run learning with a mixture of normal & binary
% note, we ran compass_em once with the default setting for parameters and then ran it a second time with
% the resulting learned parameters.
[XSmt,SSmt,Param,XPos,SPos,ML,YP,YB]=compass_em([1 1],Uk,ln,lb,Yn,Yb,Param,Valid);

%% Deviance analysis
[DEV_C,DEV_D]= compass_deviance([1 1],ln,lb,Yn,Yb,Param,Valid,XSmt,SSmt);
```

compass_param_covariance_info

This function returns the covariance matrix estimates for model free parameters. The covariance estimate can be used to identify the confidence interval of the model parameters. This function is called after [compass_em](#) and it is used to examine significance of the model being built for the data. The covariance estimate of the model parameters can be used to refine the model structure and to check whether each element is significant or not. Parameters whose variance (on the diagonal) implies a distribution that does not include zero could be considered significant. Parameters with a wide variance (distribution covers zero) should be considered for removal from the model. Covariance analysis is similar to GLM covariance analysis (Reference 7 – Ch10). It can be used to assess goodness of the model and to refine the model structure.

Syntax

```
[COV_X,COV_C,COV_D]=compass_param_covariance_info(DISTR,Uk,ln,lb,Yn,Yb,Param,obs_valid,XSmt,SSmt)
```

Description

```
[COV_X,COV_C,COV_D]=compass_param_covariance_info(DISTR,Uk,ln,lb,Yn,Yb,Param,obs_valid,XSmt,SSmt)
```

This function gets all the input being passed to [compass_em](#) and the filter result returned by [compass_em](#): [XSmt](#) and [SSmt](#). Note that in this function, we assume that all model parameters are known, and that we have an optimal estimate of the state variables. In other words, we use maximum likelihood estimates of the model parameters and state variables here. It returns the covariance estimates for the state transition process free parameters and similarly for free parameters of the continuous and discrete observation processes. The function returns three structures:

- 1) [COV_X](#): which carries covariance matrix estimate for the parameters of the state-transition process. Note that we estimate the noise variance as well.
- 2) [COV_C](#): which carries covariance matrix estimates for parameters of the continuous observation process. We estimate the noise variance or dispersion depending on the distribution.
- 3) [COV_D](#): which carries covariance matrix estimates for parameters of the discrete observation process.

Input Arguments

DISTR	<p>A 2 element vector which defines the type of the observed signals and their corresponding distributions.</p> <p>DISTR=[1 0] means there is only continuous observation signal and the proper distribution for the signal is Normal. Note that the input argument <code>Yn</code> carries the continuous signal.</p> <p>DISTR=[2 0] means there is only continuous observation signal and the proper distribution of the signal is Gamma. Note that the input argument <code>Yn</code> carries the continuous signal.</p> <p>DISTR=[0 1] means there is only discrete observation – a Bernoulli observation process. Note that the input argument <code>Yb</code> carries the continuous signal.</p> <p>DISTR=[1 1] means there are both continuous and discrete observation signals and the proper distribution for the signal is a mixture of Normal and Bernoulli distributions. Note that the input arguments <code>Yn</code> and <code>Yb</code> carry the mixed signals.</p>
-------	--

	<p>DISTR=[2 1] means there are both continuous and discrete observation signals and the proper distribution for the signal is a mixture of Gamma and Bernoulli distributions. Note that the input arguments <i>Yn</i> and <i>Yb</i> carry the mixed signals.</p> <p>Example: <i>DISTR</i> = [1 0], it means the observed signal includes only continuous observation. Note that we might define the state-space model with both continuous and discrete observations; the EM algorithm might process either continuous or discrete parts of the process given the <i>DISTR</i> elements.</p>
Uk	<p>A matrix of size $K \times nUk$ which defines the input to the state-transition model – K is the number of observations, and nUk is number of elements per each time index. The state-transition process is defined by:</p> $X(k+1) = A X(k) + B U(k) + W$ <p>Example: <i>Uk</i> = <i>ones</i>(100,1), it mean that input is a fixed value of 1 at each time index. Note that it is assumed that K is equal to 100, which is the length of the observation.</p>
In	<p>A matrix of size $K \times nIn$ which defines the input to the continuous observation process. For further information about how columns of <i>In</i> are utilized in the continuous observation process, check the compass_create_state_space function.</p> <p>Example: <i>In</i> = <i>randn</i>(100,5); it means a random input with 5 elements at each time index is passed to the continuous observation process. Note that it is assumed that K is equal to 100, which is the length of the observation.</p> <p>Practically, the <i>In</i> elements could be indicator functions of the task factors or even continuous values, including history terms.</p> <p>Note that <i>Uk</i>, <i>In</i>, and <i>Ib</i> share the same number of rows.</p>
Ib	<p>A matrix of size $K \times nIb$ which defines the input to the discrete observation process. It is equivalent to <i>In</i> of the continuous part.</p> <p>Example: <i>Ib</i> = <i>randn</i>(100,5); it means a random input with 5 elements at each time index is passed to the discrete observation process. Note that it is assumed that K is equal to 100, which is the length of the observation.</p> <p>Practically, the <i>Ib</i> elements could be indicator functions of the task factors or even continuous values, including history terms.</p> <p>Note that <i>Uk</i>, <i>In</i>, and <i>Ib</i> share the same number of rows.</p>
Yn	<p>A vector of size $K \times 1$ which carries the continuous observation signal - K is the length of observation process.</p> <p>Values of <i>Yn</i> can be real values or NaN. Note that for a NaN observation signal, the corresponding <i>obs_valid</i> must be either 0 or 2. Check <i>obs_valid</i> for further information.</p> <p>Note that for an observation process with a Normal distribution, <i>DISTR</i> = [1 <i>X</i>], the <i>Yn</i> elements might be real values. For the observation process with a Gamma distribution, <i>DISTR</i> = [2 <i>X</i>], the <i>Yn</i> elements might be positive real-valued.</p>

	<p>Example: $Y_n = rand(100,1)$; it means a positive random input with 100 rows.</p> <p>Note that U_k, I_n, I_b, and Y_n share the same number of rows.</p>
Yb	<p>A vector of size $K \times 1$ which carries the discrete observation signal - K is the length of observation process.</p> <p>Values of Yb are 0, 1, or NaN. Note that for a NaN observation signal, the corresponding obs_valid must be either 0 or 2. Check obs_valid for further information.</p> <p>Value of 0 means an incorrect response, and a value of 1 means a correct response.</p> <p>Example: $Yb = randi(2, [100\ 1]) - 1$, which means a vector with random values of 0 or 1 at each row.</p> <p>Note that U_k, I_n, I_b, Y_n, and Yb share the same number of rows.</p>
Param	<p>A structure array consisting of both the state-space model structure plus EM learning parameters.</p> <p>Check Param in compass_create_state_space and compass_set_learning_param for a more complete description.</p>
obs_valid	<p>A vector of length K with elements of 0, 1, and 2.</p> <p>An element with value 1 means the observed signal is valid. An element with value 0 means the observation is missing. An element with value 2 means the observation is censored. The censoring threshold is defined by the Param.censor_time.</p> <p>Note that a NaN value in either Y_n or Y_b must be accompanied by a value 0 or 2 in obs_valid, otherwise the EM algorithm fails to have a correct estimate of the model parameters and state variables.</p> <p>Note that obs_valid determines how the EM algorithm treats each row of the observed signal in running either E- or M-step of the EM process. Thus, setting correct values of the obs_valid has a significant importance in running the model correctly.</p> <p>Example: $obs_valid = ones(100,1)$, it means all elements of the observed signals are filled with valid values.</p> <p>Note that U_k, I_n, I_b, Y_n, Y_b, and obs_valid share the same number of rows.</p>
XSmt	<p>It is similar to XSmt defined in compass_em output argument. Check compass_em for further information</p>
SSmt	<p>It is similar to SSmt defined in compass_em output argument. Check compass_em for further information</p>

Output Arguments

COV_X	<p>This structure returns covariance matrices for parameters of the state-transition process.</p> <p><code>r = 1:nx</code> <code>COV_X{r}.A</code>: covariance matrix for matrix A, r-th row <code>COV_X{r}.AB</code>: cross covariance matrix for matrix A & B, r-th row <code>COV_X{r}.B</code>: cross covariance matrix for matrix B, r-th row <code>COV_X{r}.W</code>: variance estimate for r-th state variable model <code>COV_X{r}.SE_W</code>: std error for r-th state variable model</p> <p>If we have <code>nx</code> state variables, each of these <code>nx</code> structures will address the covariance estimate for the corresponding state variable.</p>
COV_C	<p>This structure returns covariance matrices for parameters of the continuous observation process</p> <p><code>COV_C.C</code>: covariance matrix for C – check observation model definition <code>COV_C.SE_C</code>: std error for C – check observation model definition <code>COV_C.CD</code>: covariance matrix for C & D parameters – check observation model definition <code>COV_C.D</code>: covariance matrix for D parameters – check observation model definition <code>COV_C.SE_D</code>: std error for D – check observation model definition <code>COV_C.V</code>: variance for observation process noise</p>
COV_D	<p>This structure returns covariance matrices for parameters of the discrete observation process</p> <p><code>COV_D.E</code>: covariance matrix for E – check observation model definition <code>COV_D.SE_E</code>: std error for E – check observation model definition <code>COV_D.EF</code>: covariance matrix for E & F – check observation model definition <code>COV_D.F</code>: covariance matrix for F – check observation model definition <code>COV_D.SE_F</code>: std error for F – check observation model definition</p>

Note that the function also returns covariance matrices estimates for model fixed parameters – the subset of parameters being excluded from training step. These are found in the corresponding rows of their matrices. This will be helpful for model modification and provides a better sense about whether the choice of those parameters was reasonable.

Example

Here, we call `compass_param_covariance_info` after `compass_em`. We can check output arguments of the function to find significant components.

```
%% Run learning with a mixture of normal & binary
% note, we ran compass_em once with default setting for parameters and then
% set parameters close to estimated one.
[XSmt,SSmt,Param,XPos,SPos,ML,YP,YB]=compass_em([1 1],Uk,ln,lb,Yn,Yb,Param,Valid);
```

%% Covariance analysis

```
[COV_X,COV_C,COV_D]=compass_param_covariance_info([1 1],Uk,In,Ib,Yn,Yb,Param,Valid,XSmt,SSmt);
```