v0.91

Roland Pfister

# CORE – Manual

Formerly: Sequence Generator


## Abstract

**Constraint Randomization Environment (CORE) is a MATLAB-based program for generating pseudorandom numeric sequences that follow specific constraints. It is optimized for factorial designs and includes design specifications as well as numerous tools for setting user-defined constraints.**


# Table of Contents


1. **Program Structure** ..... 2
2. **Factors (Step 1)** ..... 3
3. **Blocks (Step 2)** ..... 4
   - Trial Types ..... 4
   - Blocks ..... 5
4. **Constraints (Step 3)** ..... 5
   - Transition specification ..... 5
   - Intra- & inter-factor transitions ..... 5
5. **Play (Step 4)** ..... 7
   - Run Length Constraints ..... 7
   - Optimization Settings ..... 7
   - Play and Progress Plot ..... 8
   - Batch Processing ..... 9
6. **Evaluate (Step 5)** ..... 10
   - Summary Statistics ..... 10
   - Output Options ..... 11
7. **Algorithm and Modifications** ..... 12
   - Algorithm: Basics ..... 12
   - Code Structure ..... 13
   - Modifying Saved States ..... 15
8. **Settings** ..... 16
9. **Remarks** ..... 17
   - Future Directions ..... 17
   - Copyright ..... 17

# 1. Program Structure

The Constraint Randomization Environment **CORE** provides a modular interface, separating different steps in sequence generation (**Figure 1**). A control panel to the right side of the interface (**Steps**) guides through the generation process while a second panel (**Options**) will allow program configuration in future releases of CORE. In release 0.91, a Quit button is used as dummy button. The icons in the top row of the interface can be used to start a new project, save the current project or load previously saved projects.

First, the manual will now present detailed information for each step in the process:

- **Factors**: Design specification
- **Blocks**: Trial list specification
- **Constraints**: Transition settings
- **Play**: Generate sequences
- **Evaluate**: View and export sequences

Following these topics, the manual will explain how to modify the code for maximum customization or implementation in other MATLAB scripts and applications.

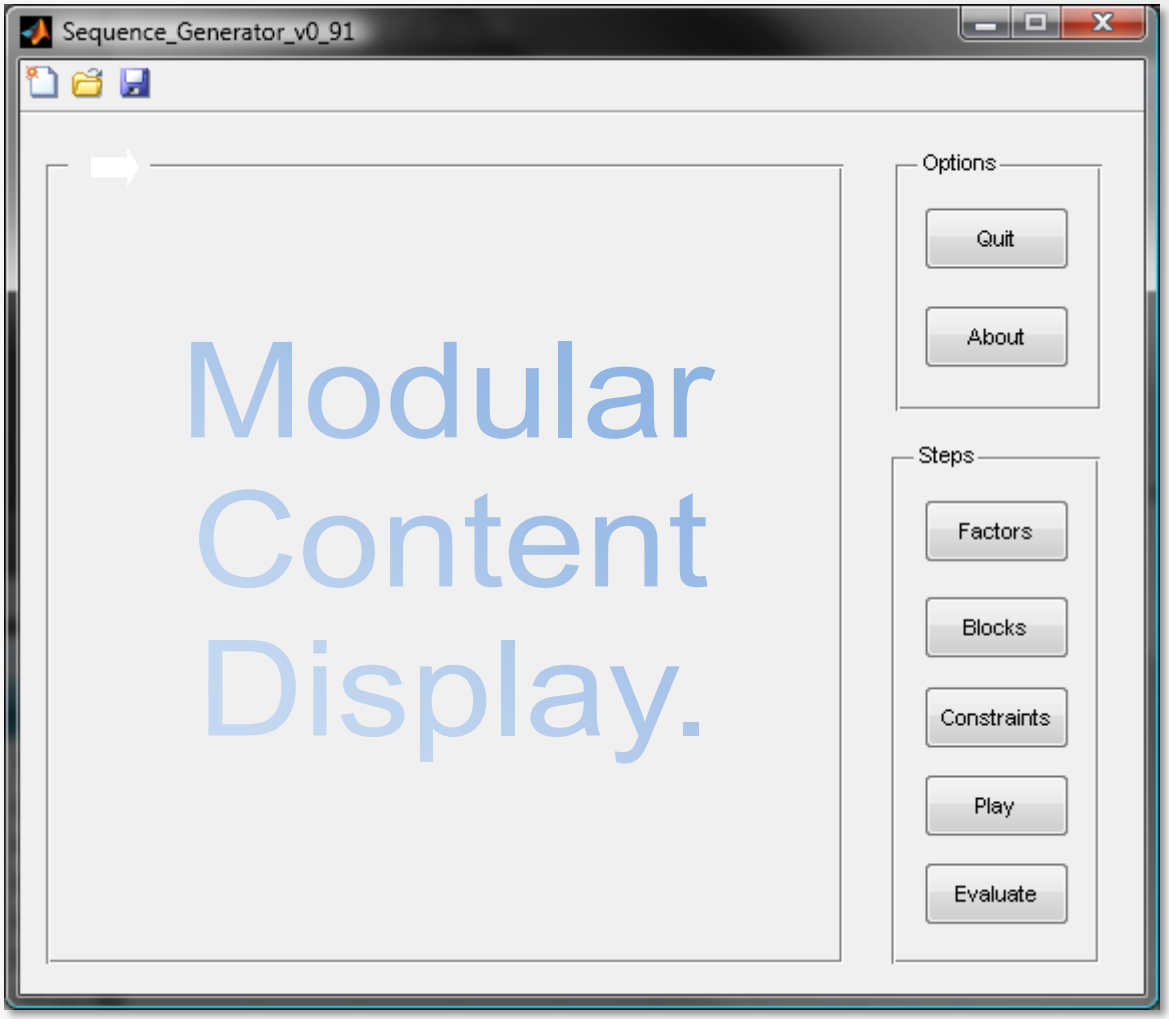



**Figure 1.** General interface of CORE. The control panels to the right (Options and Steps) can be used to navigate through the program. When starting up, the Factors panel is displayed by default.

# 2. Factors (Step 1)

The Factors panel contains the most basic information about the experimental design. It is based on the logic of factorial ANOVA and allows several factors with distinct levels to be specified. The number of factors can be adjusted with the +/- buttons below the factors table (**Figure 2**). Any number of factors and any number of levels can be used – large numbers, however, may result in serious performance losses.

Custom factor labels and level labels can be used. These names are only used inside the CORE interface and will not be included in the output (what might become a feature of a future release). Custom labels are still useful to avoid confusion in further steps of the sequence generation procedure.

Note that factor settings influence all following steps in sequence generation.

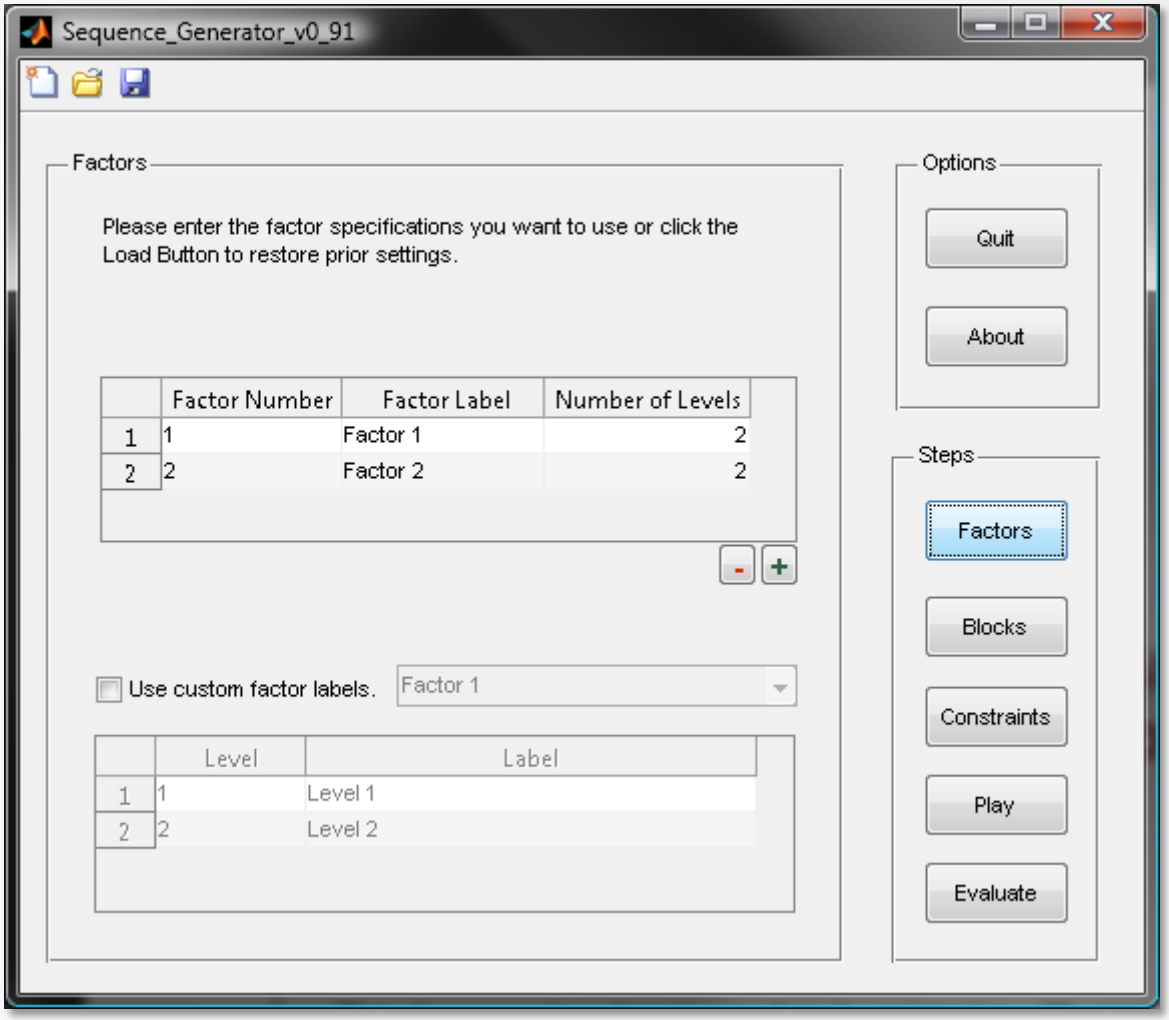



**Figure 2.** The Factors panel is the first step in sequence generation. Based on the logic of factorial ANOVA, several factors with distinct levels can be specified.

# 3. Blocks (Step 2)

Like the Factors panel, the Blocks panel is also used for design specification. This time, however, the number of to be generated elements can be edited (**Figure 3**). To allow maximum customizability, CORE introduces a new parameter called “trial type”.

## Trial Types

Each trial type represents a specific combination of factor levels. A typical 2 x 2 design thus results in 4 trial types as shown in **Figure 3**.

There are two ways of adjusting the number of trials. First, if your design includes equal numbers of trials of each trial type, simply enter this number in the text box next to “Number of Trials” and press Enter. CORE then automatically fills each row of the Count column with this number. The total number of trial is displayed in the right text box.

The second way of defining the number of trials is editing the Count value for each trial type individually. Using the first method overwrites these custom settings.

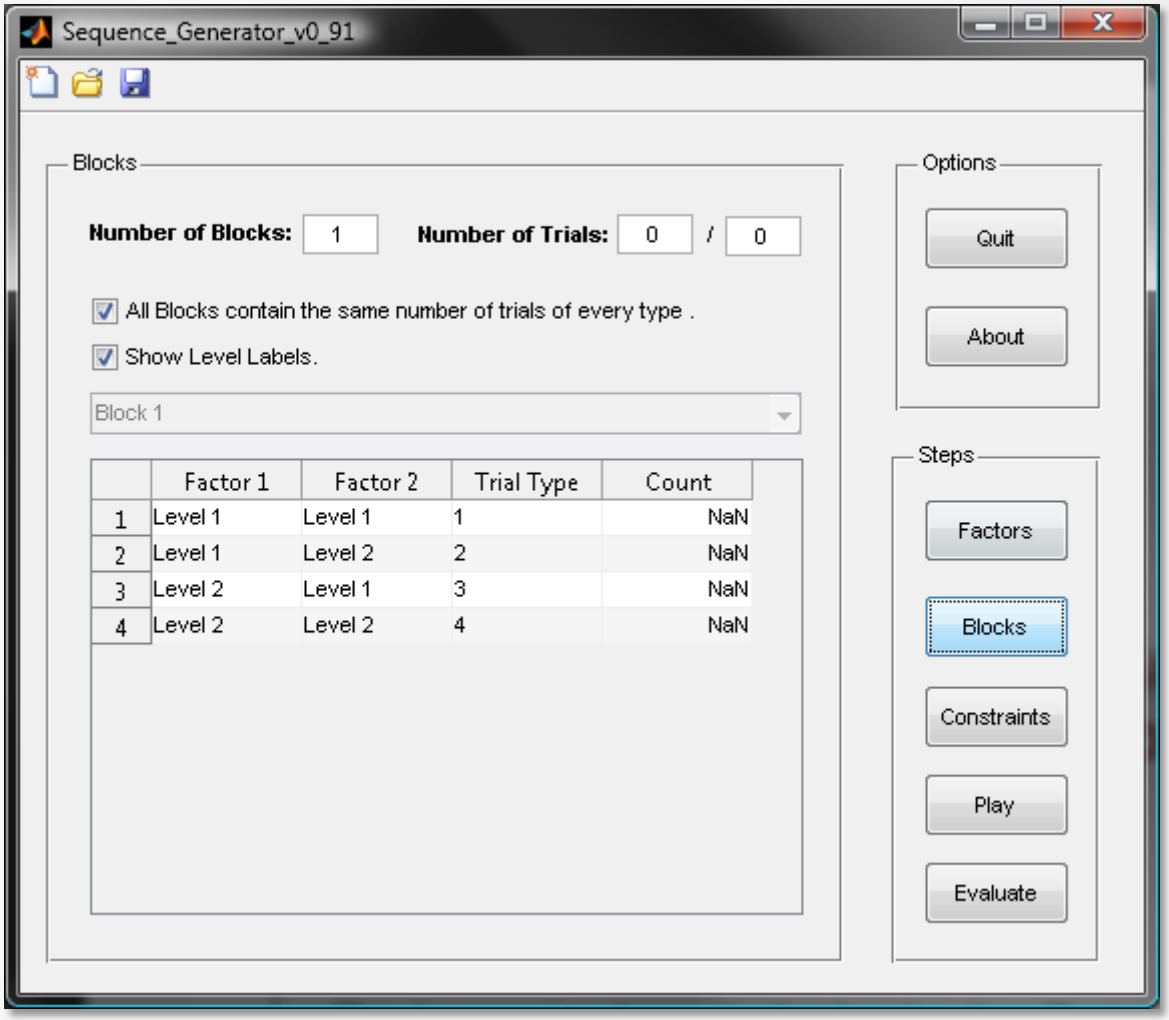



**Figure 3.** The Blocks panel is used to adjust the number of trials in the experiment. Trials can be clustered in blocks that may consist of different trial sets.

### Blocks

The number of blocks can be used to generate several instances of the trial list (corresponding to the concept of experimental blocks). By default, CORE uses the same set of trials for every block but it is also possible to specify different sets of trials for different blocks.

However, all blocks have to contain an equal number of trials – only the relative proportion of different trial types can be varied. If this requirement is violated, the total trial count turns red and no sequence will be generated.

## 4. Constraints (Step 3)

Four types of constraints can be specified: intra-factor transitions, inter-factor transitions, run length of trial types (Play panel) and run length of factor levels (Play panel). Online help for the two constraints to be specified at this step is provided via the button to the top right of the window (**Figure 4**), the X button can be used to reset transition settings to equal numbers that sum to the total number of trials in the design.

### Transition specification

In the current release of CORE, transitions are specified globally, i.e. the number of transition constraints refers to all blocks of the experiment. As only in-block transitions are evaluated, the total number of transitions is equal to the total number of trials minus the number of blocks (there are only n-1 transitions for each block of n trials).

The number of specified transitions is displayed at the top of the panel (left text box; only information of the last table edited is displayed) and should be about as big as the total number of transitions (right text box). CORE can, however, deal with deviations from the optimum value what can be customized in the Play panel.

### Intra- & inter-factor transitions

Intra-factor transitions and inter-factor transitions can be specified independently and each of them may or may not be used for sequence generation.

To illustrate the function of both transition types, we will use the 2 x 2 design that was displayed in **Figure 3**:

| Factor 1 | Factor 2 | Trial Type |
|---|---|---|
| Level 1 | Level 1 | 1 |
| Level 1 | Level 2 | 2 |
| Level 2 | Level 1 | 3 |
| Level 2 | Level 2 | 4 |

Now consider the following situation: Trial n-1 was of trial type 1 and trial n was of trial type 2. This transition can be seen as any of four cases:

- Factor 1 Level 1 is repeated (top left cell in **Figure 4**)
- Factor 2 Level 1 is followed by Factor 2 level 2
- Factor 1 Level 1 is followed by Factor 2 Level 2 – an inter-factor transition
- Factor 2 Level 1 is followed by Factor 1 Level 1

The former two cases represent intra-factor transitions whereas the latter two represent inter-factor transitions. Each has to be specified separately even though equal transition numbers are generated when the Constraints panel is opened for the first time (and can be reset at any point by pressing the X button). Resetting is also useful when trial numbers are altered after the Constraints panel was opened once.

Transition specifications will only be used for sequence generation when the box at the top left of each table is ticked. By default, however, transition numbers are still computed and can thus be used for evaluating a final sequence.

For most sequences, transition constraints will not be crucial but they provide a powerful tool for controlling even subtle aspects of a sequence.

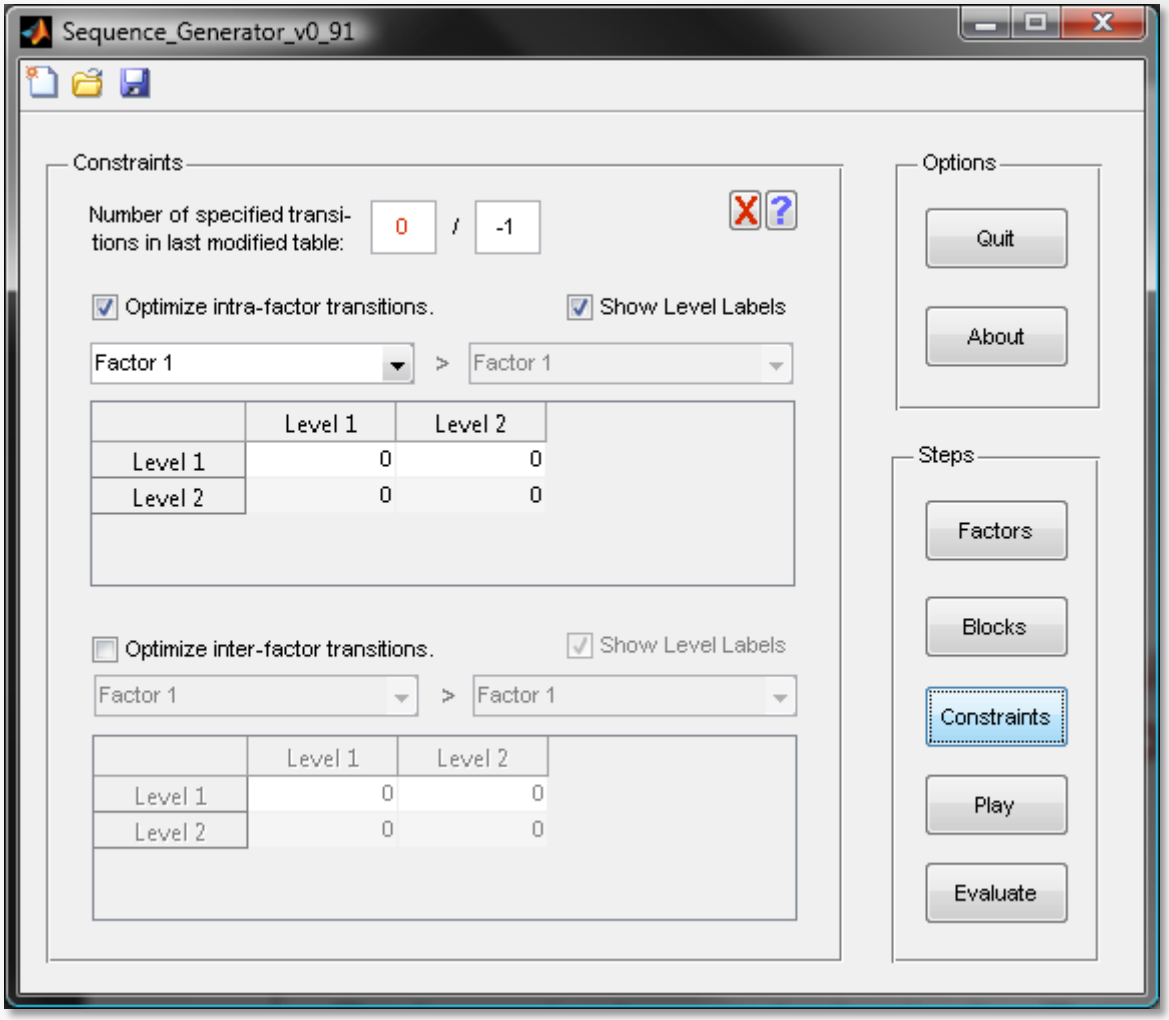



**Figure 4.** The Constraints panel adjusts the target number of intra-factor transitions and inter-factor transitions. Further constraints (run lengths for trial types and factor levels) can be set in the Play panel. The left text box contains the sum of specified transitions for intra-factor transitions (0) while the right text box indicates the overall number of transitions in the current design which is computed as the total number of specified trials minus the number of blocks. As no trials were specified in the last step (**Figure 3**), -1 transitions are displayed.

# 5. Play (Step 4)

The Play panel lets you specify further constraints and starts the generation process. Further controls include a batch processing sub-panel as well as a progress plot for visual sequence inspection. In case you want to generate a simple list of trial types without randomization, you can simply deactivate the checkbox at the top left of the panel (**Figure 5**).

## Run Length Constraints

In CORE, run length refers to the maximum number of **repetitions** that are allowed for a given trial type or factor level. The following (re-used) sequence exemplifies how both parameters are computed.

| Factor 1 | Factor 2 | Trial Type |
|---|---|---|
| Level 1 | Level 1 | 1 |
| Level 1 | Level 2 | 2 |
| Level 2 | Level 1 | 3 |
| Level 2 | Level 2 | 4 |

Assume that CORE produced the following sequence of trial types:

3 1 2 2 3 1

The maximum number of repetitions of all trial types is 1 (2→ 2) whereas the max imum number of factor level repetitions is 2 as trial type 1 and trial type 2 both trial types belong to Factor 1 Level 1.

Currently, it is not possible to allow any violations of the maximum run length / repetition settings. Also, the run length specification is treated a global parameter so that only the maximum number of repetitions across all trial types (or factor levels) will be saved. A more specific setting of run length constraints might become a feature in future releases.

Both run lengths parameters will only be used for sequence generation if the respective box is ticked. They are, however, still computed and will be displayed in the Evaluation panel once a sequence is generated.

## Optimization Settings

Three settings directly affect the behavior of CORE during the process of sequence generation: maximum iteration count, beginning threshold, and target accuracy.

The **maximum iteration count** indicates how many sequences should be tested against the specified constraints before CORE aborts the process even if no valid sequence was found. Sequence generation can also be manually aborted with the Stop button (left button of the center controls).

The **beginning threshold** indicates which minimum deviation from specified transition numbers should be used to cache the best sequence even if it does not match the criterion. The number refers to the sum of squared differences between transition specifications and transitions within the sequence that is currently evaluated.

**Target accuracy** also refers to the sum of squared differences but indicates the maximum sum that still qualifies as a valid sequence. This parameter is especially useful when not all transition constraints can be matched simultaneously (or if this case is very unlikely). A target value of 0 results in very long computation times in most cases.

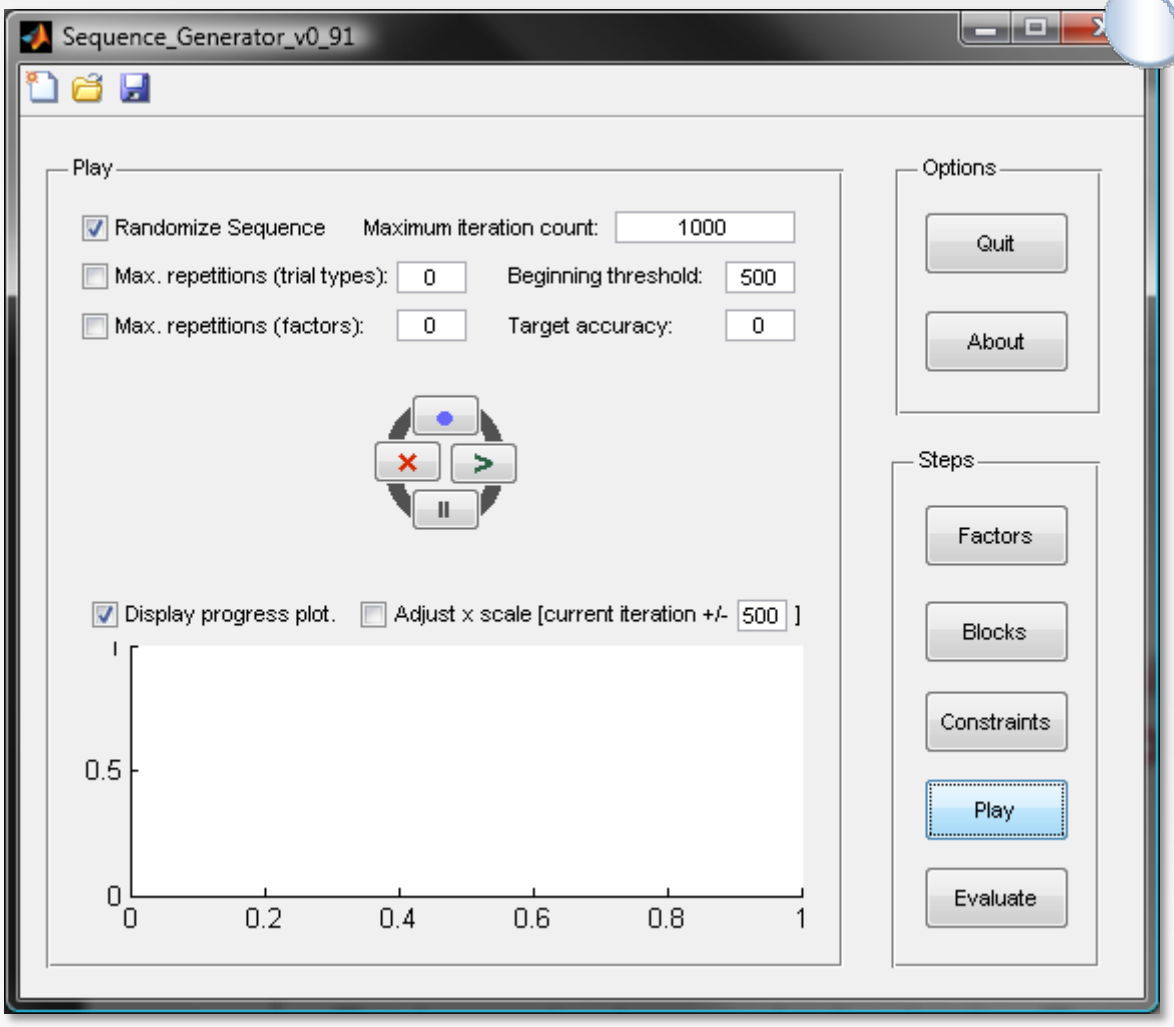



**Figure 5.** The Play panel includes further constraint specifications as well as batch processing options (top central control). After all settings have been made, the Play button starts the generation process.

## Play and Progress Plot

Pressing the Play button will start the sequence generation process. Each iteration, a random permutation of all selected trial types is created and tested against the constraints specified.

The process can be paused or aborted at any time but the current iteration will still be completed – CORE does not do things by halves.

The progress plot serves as a visual inspection tool of the generation process.

The y-axis displays the sum of squared differences between transition specifications and transition numbers in the current sequence (gray line) or the best preliminary results (blue line). The iteration index is plotted on the x-axis.

The progress plot is designed as detection tool for specification errors. However, plotting needs computation time so that it might be worth turning the tool off when no errors are apparent.

## Batch Processing

The top central control of the Play panel enables the batch processing sub-panel (**Figure 6**). Each sequence will be written to a separate output file in the chosen folder and a counter is appended to the output file name. All batch processing settings have to be completed before the generation process is started.

Batch processing output can be any of the four output types (see Evaluate (Step 5) for details):

- MATLAB (.mat)
- Text (PC; .txt)
- Text (Unix; .txt)
- MS Excel (.xls)

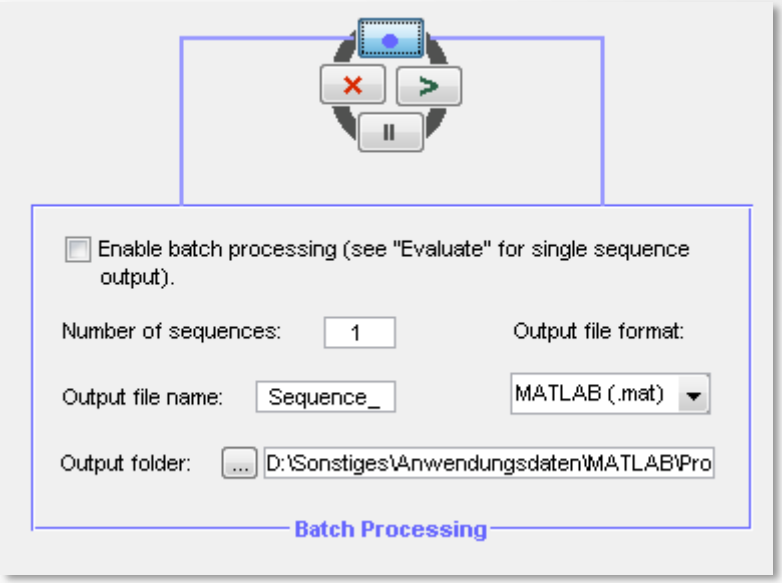



**Figure 6.** The Batch sub-panel can be accessed from the Play panel. Any number of sequences may be generated though all batch specifications have to be completed before the generation process is started. All generated sequences can be evaluated separately in the Evaluate panel.

# 6. Evaluate (Step 5)

The Evaluate panel (Figure 7) contains various statistics of generated sequences. It can only be accessed when a valid sequence was produced with the Play panel (or a preliminary sequence is cached).

## Summary Statistics

Several aspects of a sequence can be assessed. First, the sum of squared differences refers to the difference between transition specifications and transitions in the sequence. The summed squared difference relates to the summary tables of the Evaluate panel which display the number of transitions in the sequence (upper table) and the difference to the respective specifications (lower table).

The maximum repetitions statistics refer to the maximum number of trial type repetitions (left box) and factor level repetitions (right box).

If batch processing was enabled, separate results can be viewed for each sequence (top left dropdown menu).

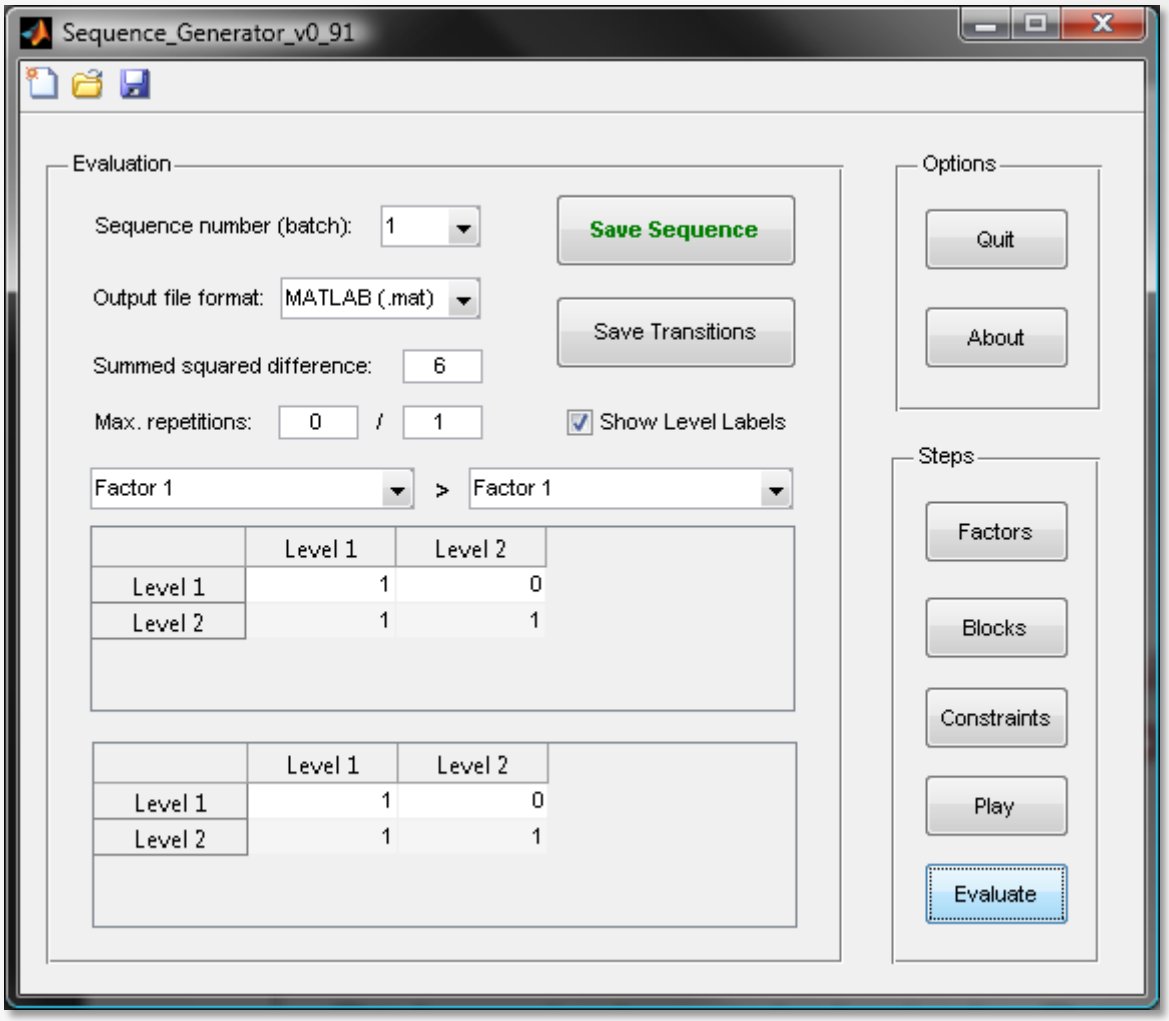



**Figure 7.** The Evaluate panel displays information about generated sequences and offers several output options.

## Output Options

Two output files can be generated for each sequence: the sequence itself and transition tables. The **sequence output** generates an output file that contains a list of trial types organized in rows while blocks are separated in columns (**Figure 8**). The structure of transition output is explained in **Figure 9**. Four output file types are possible and are explained in the figures:

- MATLAB (.mat)
- Text (PC; .txt)
- Text (Unix; .txt)
- MS Excel (.xls)

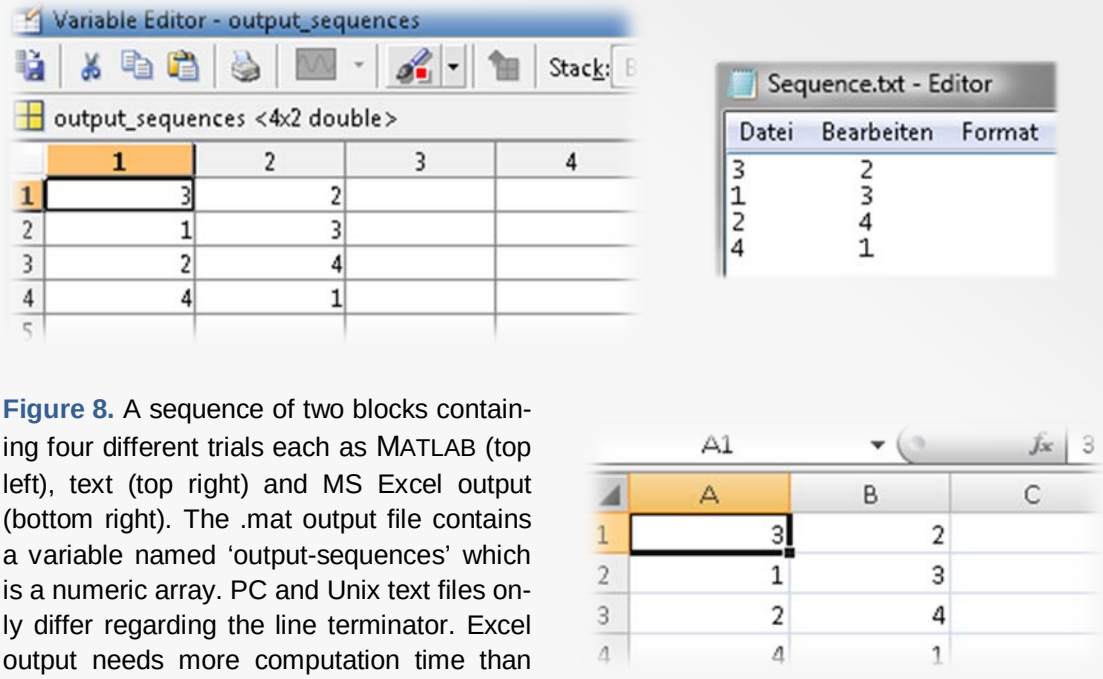
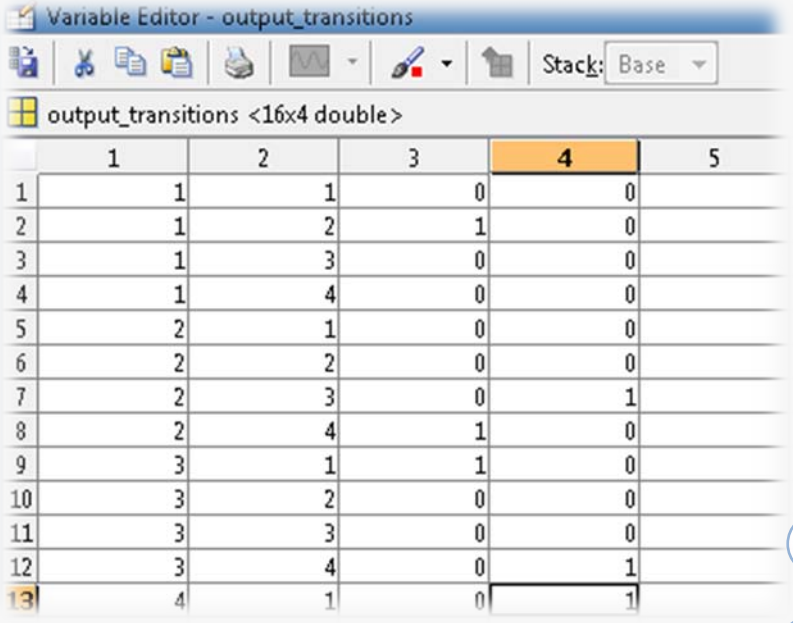



**Figure 8.** A sequence of two blocks containing four different trials each as MATLAB (top left), text (top right) and MS Excel output (bottom right). The .mat output file contains a variable named 'output-sequences' which is a numeric array. PC and Unix text files only differ regarding the line terminator. Excel output needs more computation time than other output file types.

**Figure 9.** Transition output of any file type contains at least three columns of data: the first two columns represent the trial type at trial n-1 (1st column) and at trial n (2nd column). The following columns contain the number of these transitions in each block of trials. One column per block is used.

This example is the .mat output of the sequences displayed in **Figure 8**.

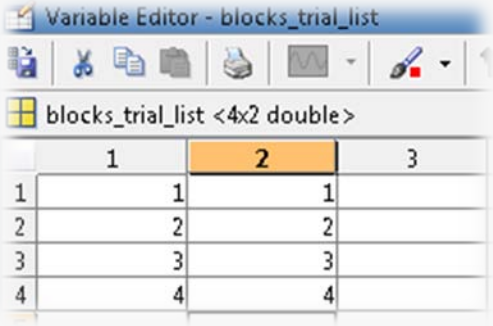


# 7. Algorithm and Modifications

## Algorithm: Basics

CORE uses a brute force algorithm to generate pseudo-randomized sequences and tests the generated sequence against all specified constraints. The most basic variable used is blocks_trial_list which contains a number or columns corresponding to the number of blocks specified and each column consists of as many rows as trials were specified. Each number represents a specific trial type.

For the example of our 2 x 2 design with 2 blocks and 4 trials each, blocks_trial_list will look like this:




| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 1 | 1 | |
| 2 | 2 | 2 | |
| 3 | 3 | 3 | |
| 4 | 4 | 4 | |

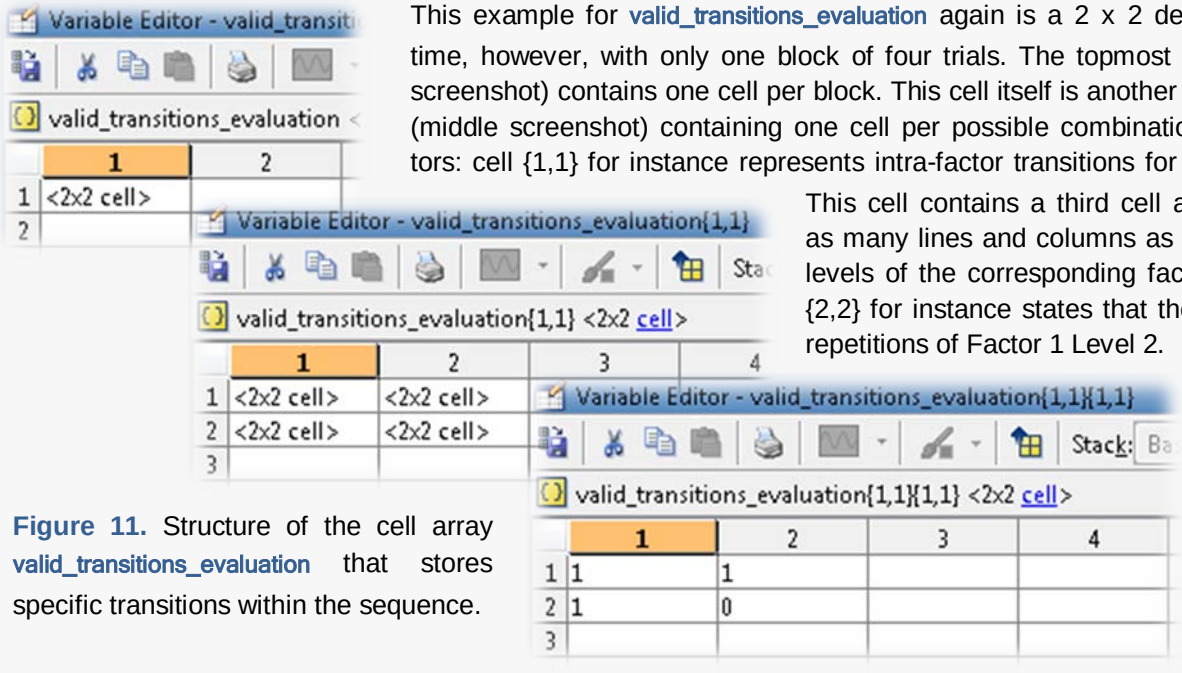
**Figure 10.** Starting point of the generation process.

In a first step, each block is randomized independently by means of the *randperm* command. The randomized sequence is stored in a temporal variable.

Then, CORE works through the sequence, evaluating the run length of every trial type and every factor level. After that, the number of transitions is computed for every possible pair of trial types.

Finally, the summed squared differences for all transitions as well as both run length specifications are compared with the stop criterion. If all criteria are met, these three variables are stored within the cell array valid_transition_parameters. Specific transitions in the sequence and are stored in another, hierarchically structured cell array (valid_transitions_evaluation). This latter array can become a surprisingly complex structure when many factors and/or many factor labels are used (see **Figure 11** for a detailed description).

This example for valid_transitions_evaluation again is a 2 x 2 design, this time, however, with only one block of four trials. The topmost level (left screenshot) contains one cell per block. This cell itself is another cell array (middle screenshot) containing one cell per possible combination of factors: cell {1,1} for instance represents intra-factor transitions for Factor 1. This cell contains a third cell array with as many lines and columns as there are levels of the corresponding factors. Cell {2,2} for instance states that there are 0 repetitions of Factor 1 Level 2.




| | 1 | 2 |
|---|---|---|
| 1 | <2x2 cell> | |
| 2 | | |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | <2x2 cell> | <2x2 cell> | | |
| 2 | <2x2 cell> | <2x2 cell> | | |
| 3 | | | | |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 1 | 1 | | |
| 2 | 1 | 0 | | |
| 3 | | | | |

**Figure 11.** Structure of the cell array valid_transitions_evaluation that stores specific transitions within the sequence.

## Code Structure

Both algorithms for **run length/repetitions** testing – one for trial type repetitions, one for factor level repetitions, follow the same structure so that only the former one is presented here.

The algorithm for trial type repetitions first gets the number of blocks and number of trials per block (**Code 1**; lines 5 and 6). Then, it iterates through all trials, starting with the second trial in each block.

For each trial, it is checked whether the preceding trial was of the same trial type or of a different trial type. If trial types are equal, a repetitions counter is increased by 1 (line 8) and compared to the current maximum number of repetitions. If both trial types are not equal, however, the repetitions counter is set to zero (line 12).

The maximum count is saved for evaluation in a separate variable.

```
1  %Run length for trial types.
2  n_repetitions_ttype = 0;
3  max_repetitions_ttype = 0;
4
5  for j = 1:size(sequence_array,2)    % Iterate over all blocks
6  for i = 2:size(sequence_array,1)    % Iterate over all trials
7      if sequence_array(i,j) == sequence_array(i-1,j)    %If trial type
8          n_repetitions_ttype = n_repetitions_ttype + 1; % is repeated:
0          max_repetitions_ttype = ...                    % > counter++1
10         max(max_repetitions_ttype,n_repetitions_ttype);
11     else
12         n_repetitions_ttype = 0;                       % if not rep.:
13     end;                                               % > counter=0
14 end;
15 end;
```

**Code 1.** Run length / maximum repetitions computation for trial types. The algorithm for factor levels has the same structure but uses a different array containing the factor levels belonging to each trial type.

The algorithm for **transition evaluation** is based on the array described in **Figure 9** which contains an additional column with the sum of transitions for each pair of trial types. This array is created in the following way (**Code 2**). [Continued on page 15]

```
1   transitions = zeros(size(possible_transitions,1),n_blocks+3);
2   transitions(:,1:2) = possible_transitions;
3   for i = 1:(size(sequence_array,1)-1)
4       for j = 1:n_blocks
5       transitions((((sequence_array(i,j)-1)*n_trial_types) + ...
6           sequence_array(i+1,j)),j+2) = ...
7       transitions((((sequence_array(i,j)-1)*n_trial_types) + ...
8           sequence_array(i+1,j)),j+2)+1;
9       end;
10  end;
11
12  for i = 1:(n_trial_types * n_trial_types)
13     transitions(i,end) = sum(transitions(i,3:(end-1)));
14  end;
```

**Code 2.** Transition array. Only the last column created in line 13 is used for further computations.

```
%Evaluate transitions for every pair of factors.
overall_difference = 0;
overall_difference_intra = 0;
overall_difference_inter = 0;

for factor_n1 = 1:n_factors
for factor_n2 = 1:n_factors

%Get current factor labels.
cf1 = ttypes_to_flevels(:,factor_n1);
cf2 = ttypes_to_flevels(:,factor_n2);

%Evaluate transitions for every pair of levels.
for level_n1 = 1:factor_list{factor_n1,3}
for level_n2 = 1:factor_list{factor_n2,3}
    ttypes_1 = ttypes(cf1==level_n1);
    ttypes_2 = ttypes(cf2==level_n2);

    for i = 1:(n_trial_types * n_trial_types)
    if sum(ttypes_1 == transitions(i,1)) == 1
        indexer_temp(i,1) = 1;
    else
        indexer_temp(i,1) = 0;
    end;
    if sum(ttypes_2 == transitions(i,2)) == 1
        indexer_temp(i,2) = 1;
    else
        indexer_temp(i,2) = 0;
    end;
    indexer(i,1) = min(indexer_temp(i,:));
    end;
    evaluation = transitions(:,end);
    indexer = logical(indexer);
    transitions_evaluation{factor_n1,factor_n2}{level_n1,level_n2}...
        = sum(evaluation(indexer));
    transitions_difference{factor_n1,factor_n2}{level_n1,level_n2}...
        = sum(evaluation(indexer)) - constraints_transitions ...
        {factor_n1,factor_n2}{level_n1,level_n2};
    if factor_n1 == factor_n2
        overall_difference_intra = overall_difference_intra + ...
            (sum(evaluation(indexer)) - constraints_transitions ...
            {factor_n1,factor_n2}{level_n1,level_n2})^2;
    else
        overall_difference_inter = overall_difference_inter + ...
            (sum(evaluation(indexer)) - constraints_transitions ...
            {factor_n1,factor_n2}{level_n1,level_n2})^2;
    end;

end;
end;

end;
end;
```

**Code 3.** Transition computation for intra- and inter-factor transitions. Sum of squares for both transition types are later combined to an overall sum of squares (depending on user specifications). See text for a more detailed description.

Using a further array containing the mapping of trial types to factor levels, the transition array is restructured so that is corresponds to the cell array described in (**Figure 11**, valid_transitions_evaluation). Then, both arrays are compared element-wise while the differences for each cell are squared and summed to arrive at a measure of fit between transitions in the sequence and user-specified constraints.

Especially the transition evaluation takes a lot of computation time so that it is highly recommended to create a modified version of this code for more complex applications.

## Modifying Saved States

CORE-sessions can be saved as .mat files. These files contain the single structure Saved_Sequence. The structure has 3 fields (**Figure 12**) that can be modified without running CORE.

**General_Information** is a string indicating the version of CORE ("Sequence Generator" up to version 0.91, "CORE" for all later releases).

**Program_Settings** stores settings for all elements of the CORE interface. Tables are stored as cell arrays, all other elements either as integers or strings (depending on typical MATLAB commands for the handles structure of GUIDE).

**Base_Workspace** contains a number of variables that are normally stored in the base workspace of MATLAB, including all that are relevant for evaluation and output.

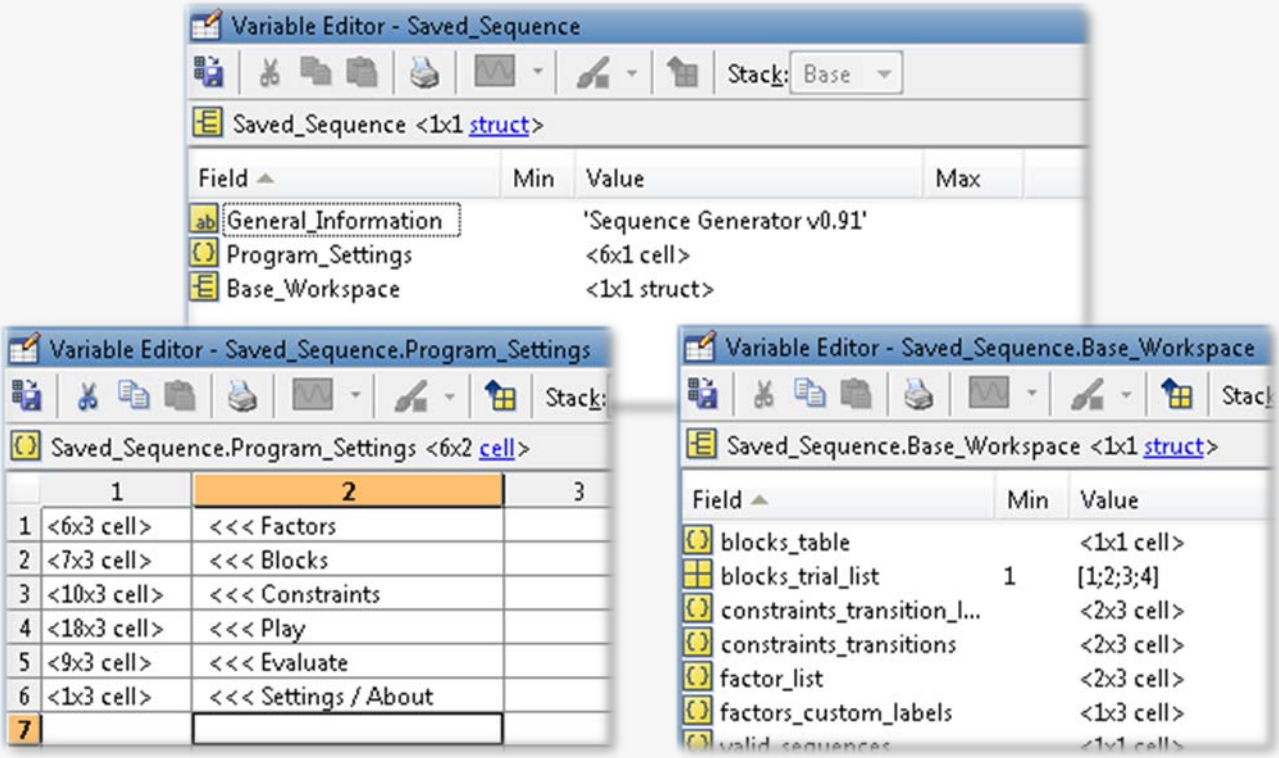



**Figure 12.** Structure of a save file (.mat). See text for a more detailed description.

# 8. Settings

To be introduced in future releases of CORE….

# 9. Remarks

## Future Directions

The following features are currently discussed as features in future releases of CORE:

- Settings panel (including options for detailed control over what is computed (even if not displayed))
- Experiment/project name in Factors panel.
- Run length (repetition) settings for each trial type and factor level individually (optionally) – will require a redesign of the run length interface which will be included in the Constraints panel
- Redesign of the Constraints panel so that it provides the possibility to switch between transition and run length constraints
- Adjusting the sequence generation mechanism to allow unequal blocks
- Adding run length and maximum run length to the progress plot
- New optional algorithm ('half-brute') allowing faster convergence on working solutions
- Output: Optional replacement of trial type numbers with a list/table of factor level labels
- The plot might be removed in the next release and will be substituted by another diagnostic tool
- Transition specifications on the level of individual trial types

## Copyright