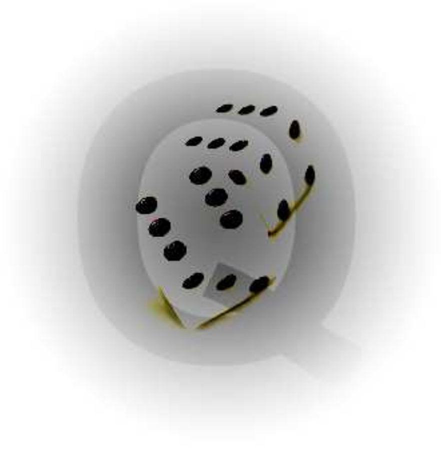
RQUBE

–

the manual



by Jan Seifert and Patrick Britz

Contents

List of Figures

System requirements

- Operating system: Microsoft Windows 95/98/ME/2000/XP and later
- 80486 Processor or later (the faster, the better)
- $\geq$ 8MB RAM
- $\geq$ 5MB of available hard-disk space
- Sound card (if you want too use the Beecrypt RNG)

Many thanks to...

- Dominik Reichl for providing the CTrueRandom class that gives RQube access to Random.org (see www.dominik-reichl.de).
- Jasper Bedaux for his C++ port of the Mersenne Twister (see bedaux.net).
- Everyone in the EEG laboratory at the university in Trier. Without their guidance and support neither of us would be where he is now.
- ...everybody for helping us.

Chapter 1

What is RQube?

1.1 Introduction

Randomisation is a basic procedure in experimental design. Considering the importance of it, it is strange to say, that – except for a few articles in parapsychological research – the topic did not receive much attention.

> "Every finite symbol string is random under some nontrivial definitions of randomness.
> Every finite symbol string is nonrandom under some nontrivial definitions of randomness."

These paradox conclusions in Gilmore (1989, p. 339) underline the fact, that it might be worth the effort to consider the reasons of randomisation and the principles of the process.

1.1.1 Why Randomise Trial Sequences?

There are several reasons to shuffle the items in a trial series. Not all of them can be satisfied at once. Not all of them are based on the same notion of randomness.

1. Minimise anticipation processes, i.e. minimise predictability.

2. Minimise habituation effects.

3. Minimise sequence effects.

Predictability

If subjects are able to predict the upcoming event, the relevant processing is not elicited by the stimulus, but takes place before its presentation.

The common notion of randomness is based on the generating procedure, like tossing coins. But if you toss a coin and you get nothing but tails (TTTTTTT), you never would consider this coin to be fair. Nevertheless stochastic tells us, that this event is as probable as any other event ($P(\text{TTTTTTT}) = P(\text{THHTHTT})$). "[...] it should be no more surprising to obtain the series with an obvious pattern than to obtain the one that seems to be random" (Chaitin, 1975, p. 47). However, it is. The stochastic concept of randomness does not fit the intuitive perception of it (Falk & Konold, 1998; Bar-Hillel & Wagenaar, 1991). Furthermore, this series is (indeed) predictable.

Hence, there was a branch developed of mathematics which abandoned a process based definition of randomness. It was developed from research on complexity and it emphasises unpredictability as a key quality for a random sequence (Chaitin, 1975; Gammermann & Vovk, 1999). The less orderly the sequence is, the greater is its complexity and the less predictable is any element of it. Concerning experimental needs unpredictability often provides a more practical concept of randomness. To make a series of stimuli patternless and unpredictable is the principal reason why we randomise it. The subject must not be able to foresee the upcoming trial – it should be even hard to guess it.

Sequence Effects

Many experimental designs need to control the alternations between different experimental condition. Viewing an indifferent picture certainly makes another impression, when viewed after a very disgusting picture, than after a funny one. The appropriate countermeasure is to control the number alternations between experimental conditions.

Habituation

Habituation means that responses elicited by a stimulus (or a class of stimuli) diminish in strength with repeated exposures. Rare occurrences of and frequent alternations are necessary to prevent habituation effects.

1.2 How to Describe “Randomness”

Randomness in our means has often to be described as unpredictability. In stochastics you can toss a dice a hundred times and receive the “1” a hundred times. Each possible sequence occurs with equal probability. But in experimental psychology we do not want the same treatment to be repeated too often. Common random numbers generators that are implemented in C++, Visual Basic or in experimental software like E-Prime^{TM} often lead to long rows of equal treatment variables.

Another concept of probability comes from the mathematician Kolmogorov. Can there be “true” randomness under the laws of physics? He did not assume a sequence of a hundred “1” to be random, even if they had been generated in a way that demands such an interpretation (like tossing a dice or a coin). A sequence is considered to be random, if you cannot guess the following number. But in a sequence of “1” the successor is easy to foretell. So RQUBƎ helps you to minimise predictability as good as possible. The zero– and first-order restrictions are both qualified to minimise the predictability.

On the other hand, sometimes are effects of habituation the more sensitive problem for experimental data. The “Maximum run length” and the “Minimum distance” are especially qualified to control for habituation effects.

Finally, the first-order restrictions help to minimise repetition effects by equalising the frequencies of pairs of consecutive trials and by controlling the distribution of those pairs.

1.2.1 Global Characteristics

Maximum Run Length

The first restriction (Q_{RL}) is the number of equal consecutive items. Items are considered as equal, if they are in the same treatment level. The discrimination between items and treatments in succession is necessary, because sometimes the researcher does not want the same item to be repeated, but there is no problem to repeat the same experimental condition[1].

Q_{RL} may vary between $1 \leq Q_{\mathrm{RL}} \leq C$.

Minimum Distance

Minimum distance specifies the number of items that at least separate two equivalent items. It is only necessary in experiments that strictly need to forbid repetitions. The minimum of the Minimum Distance is one: $Q_{\mathrm{MD}} \geq 1$. It means that repetitions are valid; a value of 2 ensures that there is at least one deviant between two equivalent items, etc.

Probability of Zero Order

This characteristic (Q_0) is defined by the absolute number of occurrences of a treatment level.

The value of Q_0 is fixed to $Q_0 = L/C$.

[1]The letters C and L stand for the number of elements distributed over the sequence, and the *l*ength of the trial series, respectively.

Probability of First Order

The third criterion (Q_1) is based on transition frequencies (or probabilities of second order). It is undesirable that, for instance, item number four always follows item number two. The human brain may be adaptive enough, to detect such regularities and to accommodate to them. The homogeneity index Q_1 provided by RQUBƎ is the alpha error of a χ^2-test. If the expected (i.e. optimal) frequency of item-pairs matches the observed frequency the index results to zero and becomes greater with increasing heterogeneity. It’s maximum value is one. If you are interested in an algorithm that exactly controls transition frequencies, see Emerson and Tobias (1995).

Q_1 varies between $0 \leq Q_1 \leq 1$, the optimal value is $Q_1^* = 0.00$. But be aware that in most designs it will not be possible to reach that value, since it requires the fraction $(L-1)/C^2$ to be an integer solution.

1.2.2 Local Characteristics

Distribution of Zero-order Probability

With the fourth criterion (Q_{D0}) we can check, how homogeneously the items are distributed among the series. It is unwanted that any sequence ends with only a few alternating items. From our (the authors’) experience in creating stimulus sequences we know, that this kind of thin out is a common problem. In order to avoid that, the user can specify a limit of deviation from the optimal distribution.

The used index number Q_{D0} has a range $0 \leq Q_{\mathrm{D0}} \leq 1$ and can be interpreted as percentage value. The optimal value is $Q_{\mathrm{D0}}^* = 0.00$.

Distribution of First-order Probability

The last criterion (Q_{D1}) follows is calculated on the same basis as first-order dispersion. Accordingly, Q_{D1} varies between $0 \leq Q_{\mathrm{D1}} \leq 1$, the optimal value is $Q_{\mathrm{D1}}^* = 0.00$.

1.2.3 Further Characteristics

For the needs of the "expert randomiser" three further characteristics have been added to RQUBƎ. They are sometimes used in the literature about subjective randomness (Bar-Hillel & Wagenaar, 1991).

Autocorrelation

The autocorrelation is the expected value of the product of a random variable with a time-shifted version of itself. In other words, any value x_i of a sequence is correlated with its follower x_{i+lag}. Hence, it is a test of higher order characteristics of a quasi-random series.

RQUBƎ calculates the all autocorrelations up to a lag of 6 and uses the greatest value. Q_{auto} varies between $0 \leq Q_{auto} \leq 1$, the optimal value is $Q^*_{auto} = 0.00$.

Runs Test

The Runs test can be used to decide if a data set is from a random process. A run is a series of equivalent elements. The number of elements within one series is the length of the run. The Runs test provided by RQUBƎ is described by Mood (1940). If there are too many or too few runs in a series of events, it is not considered to be random.

The value Q_{runs} is the tail probability of an approximately normal distributed variable. The longer the sequence, the better the normal approximation. Q_{runs} varies between $0 \leq Q_{runs} \leq 1$, the optimal value is $Q^*_{runs} = 0.50$.

Standardized Entropy

The idea of randomness in information theory is quite similar to the notion of randomness as complexity. This theoretical connection is the reason why we added a standardized entropy measure to the list of RQUBƎ's probability characteristics.

The standardized entropy Q_H lies between $0 \leq Q_H \leq 1$, the optimal value is $Q_H^* = 1$.

1.3 Proposal of Default Values

Every experimental design has its own requirements and therefore it is rather difficult to propose standard values for each of the five attributes. It is necessary to reconsider the values and adapt them to the specific context. But nevertheless, a template may be useful for new users of this program. We generated hundreds of sequences while developing this program and we used RQUBƎ-sequences (and of its precursors) in several experiments.

These restrictions are partially contradictory. For example, if the maximum run length (Q_{RL}) is set to 1, then it will not be possible to reach a low Q_1 that controls the transition frequencies, because the transition of an item to itself is forbidden. It is clear, that it is not possible to be strict on *all* restrictions at once. We came to the conclusion, that in most cases the distribution restrictions (Q_{D0}, Q_{D1}) are less pivotal. At least, unless a very long trial series is needed.

Our experience allows us to set up a useful template. But this is only a proposal. Your own experiments may vary in their purpose and so may vary the sequential constraints. If you choose a very complex design they might be inappropriate, because they are too restrictive. If the constraints become too high, the number of possible solutions will be reduced to much, and RQUBƎ will no be able to find one of them in an acceptable period of time. Be careful in your choice, do not always rely on standard values, even if they may be appropriate in most cases.

1. *Maximum Run Length:* $\check{Q}_{\text{RL}} = (L-1)/C^2$,
   with C being the number (of combinations) of factor levels, and L being the length of the sequence. In most cases it is not advisable to set $Q_{\text{RL}} = 1$. The subject can be sure, that the same treatment will not be repeated. Furthermore, it seems a little odd to restrict the run length, at all. According to the

gamblers' fallacy, subjects expect short run lengths. If a sequence is restricted according to first-order probability, this value may be set to $\breve{Q}_0$, because this is the maximum possible value of Q_{RL}. The advantage of the Q_{RL} restriction is a faster computation speed of R**QUBƎ**.

However, sometimes predictability is not the main reason for randomisation. If the researcher wants to avoid effects of habituation, instead, Q_{RL} may be set close to 1.

2. *Minimum distance:* $\breve{Q}_{\text{MD}} = 1$

   Usually there will be no need to set Q_{MD} to anything else than one, which means that repetitions are possible. Use higher values if item repetition causes rapid habituation effects.

3. *Probability of zero order:* $\breve{Q}_0 = L/C$

   This criterion is fixed. It is defined by the number of combinations of factor levels (C) and the length of the sequence (L). R**QUBƎ** exactly satisfies this restriction.

4. *Probability of first order:* $\breve{Q}_1 \leq 0.20$

   The more complex the design, the less important a severely restricted may Q_1 be. We do advise against $Q_1 = 0.00$, which will certainly fail, unless the length of the list $L - 1$ can be divided by C^2 without remainder. If you are interested in an algorithm that controls transition frequencies exactly, see Emerson and Tobias (1995).

5. *Distribution of zero-order probability:* $\breve{Q}_{\text{D0}} \approx 0.30$

   It is far more challenging to find reasonable default values for the two distribution criteria than for the previous three. We already used R**QUBƎ** in a number of experiments. Based on our experience it is rarely necessary to go below 0.10. Our proposal is $\breve{Q}_{\text{D0}} \approx 0.30$; it may be rough, but adequate in most cases. If an experiment consists of several short blocks, and each block has its own RQUBE-list, you may switch distribution characteristics off.

6. *Distribution of first-order probability:* $\breve{Q}_{\mathrm{D1}} \approx 0.30$
   The more complex the design, the less important is it, to set Q_{D1} to a low value. Since Q_{D1} reflects more complex aspects of the relation between trials than the zero-order distribution (Q_{D0}) does, we assume, that the first-order distribution may be set a bit lower to a value of $\breve{Q}_{\mathrm{D1}} \approx 0.30$. If an experiment consists of several short blocks, and each block has its own RQUBE-list, you may switch distribution characteristics off.

Chapter 2

How to Use RQube

2.1 The Rationale of RQube

The rationale of RQUBƎ is similar to the rationale of the ANOVA. You may also define factors, each factors has its levels[1]. Even though the design in RQUBƎ may often differ from your final ANOVA design, both follow the same basic concept.

Three types of elements may be specified in RQUBƎ.

1. one *factor space* or *design*.

2. up to 21 *factors* with up to 2 levels each, or 5 factors with 18 levels each. The maximum number of combinations is set to $8^7 \approx 2,000,000$ (see also table 2.1.2).

3. up to 21 *level constraints* with up to 2 subsets each, or 5 constraints with up to 18 sets each.

The factor space is the hierarchical superordinate object. It consists of factors which define the characteristics of the experiment trials. In an attention task e.g.

[1]One of the aspects of the name RQUBƎ is the n-dimensional cube that is formed by the combination of all factors.

there is on factor that specifies the presence of noise. In many cases the factors correspond to experimental factors in the ANOVA design, but they may also specify further characteristics.

The factor space may be circumscribed as a special type of factor. Its levels are the level combinations of the subordinate factors. In a 3×4 design the factor space has 12 levels.

2.1.1 Level Constraints

Using level constraints it is possible to treat several levels of a factor as if they were equal. They help to accomplish more complex requirements, such as unequal trial frequencies (see example in section 2.3.3).

Figure 2.1 illustrates an example of how to make use of constraints. It shows the one factor of a simple Go/Nogo experiment. In this paradigm the subjects has to press a key as quickly as possible in one part of the trials. (S)he has to omit the reaction in another part. In order to activate a strong reaction tendency it is quite common that the subjects shall react on only one third of the trials.

The left side of figure 2.1 shows the common approach of RQUBƎ. Each cell is treated individually. This is - however - not exactly the way to gain a reasonable result. It may quite easily happen that RQUBƎ generates a sequence like this: `2 2 3 3 2 3 3 3 2 2 3 2`. RQUBƎ cannot know that level `2` and `3` share the same meaning.

Using level constraints you may create two sets. RQUBƎ treats each set exactly the same way it treats individual levels. This way you may combine the levels `2` and `3` to one new "super-set" and `1` to be the other. Finally, any restriction may be applied to these two sets.

In summary, controlling the factor prevents that too many stop trials (`1`) follow in succession. Controlling the level constraint prevents too many go trials in succession.

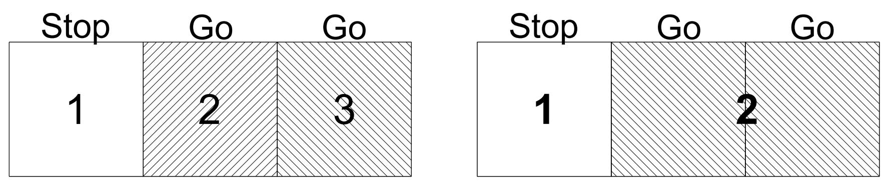


Figure 2.1: Level constraints. Left: RQube treats each level individually; right: the two go levels are handled as if they were equal.

2.1.2 Limitations

Since computer memory is limited RQUBE does also have it limits. See table 2.1.2 for the possible number of factors and levels. The more factors you design, the less levels each factor may have (on average). The maximum number of combinations is set to $8^7 \approx 2,000,000$.

Table 2.1: Maximum possible number of factors and levels (level constraints and sets, respectively).

factors	5	6	7	8	9	10	13	**21**
levels	**18**	11	8	6	5	4	3	2

2.2 The Screen Layout

The concept of the graphical user interface (GUI) is a wizard. That wizard guides you through the steps that are necessary to define a complete design and to start the computation of trial sequences. For better orientation, all steps of the wizard are based on one identical layout. This layout comprises three areas of major interest (from bottom to top):

1. the button bottom (green)

2. the working area (blue)

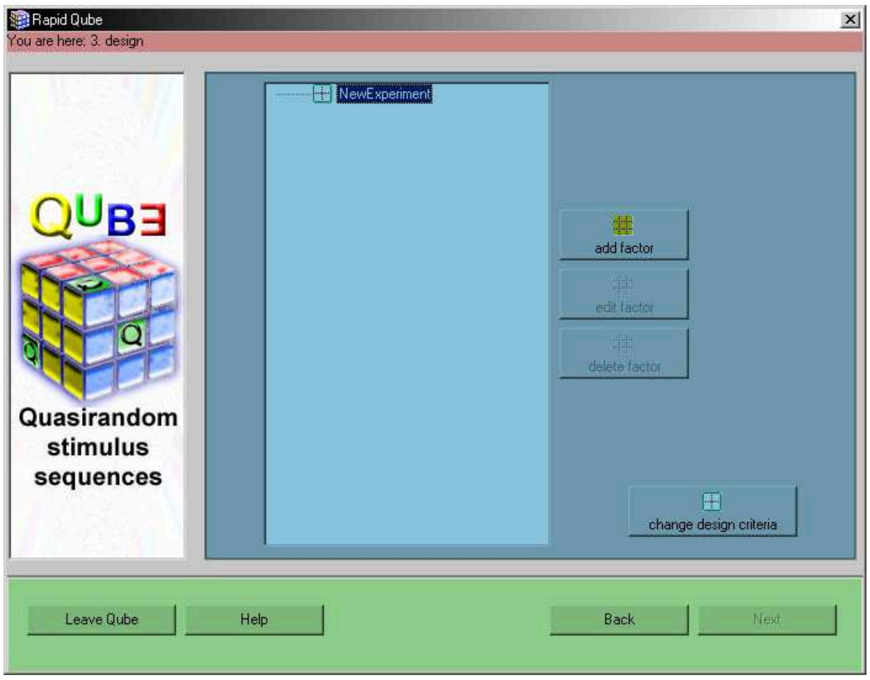


Figure 2.2: The GUI of RQUBƎ *displays three areas of major importance: (1) the navigation bar (red area); (2) the working area (blue); and (3) the button bottom (green).*

3. the navigation bar (red area)

The **button bottom** provides always the same four buttons: “Leave Qube” to shutdown RQUBƎ, “Help” to open the help file you are currently studying, “Back” to go back one single step, and “Forward” to finish the current step and move on to the next one. RQUBƎ makes sure that you may not press a button that initiates an action that is not valid in the certain context.

The **working area** contains all the necessary information the user needs to enter the data at the current step. The content of this area may change from step to step.

The **navigation bar** tells the user which step of the design the user is currently working at. It displays the number of the step (between 1 and 6) and a short description. This description has the format of a path (similarly windows describes paths: folder\folder\file). If the wizard steps into a sub-action, the step description indicates that by the “\” character. Example: one sub-step of defining the design settings is to add a new factor. This step is described by “3. Define

Design\Add Factor".

2.3 Examples

2.3.1 Emotional Slides

The first example's purpose is to explain, how the data is entered within the RQUBƎ-wizard. This example may be found at `examples\EmoPictures.qini` in your RQUBƎ installation directory.

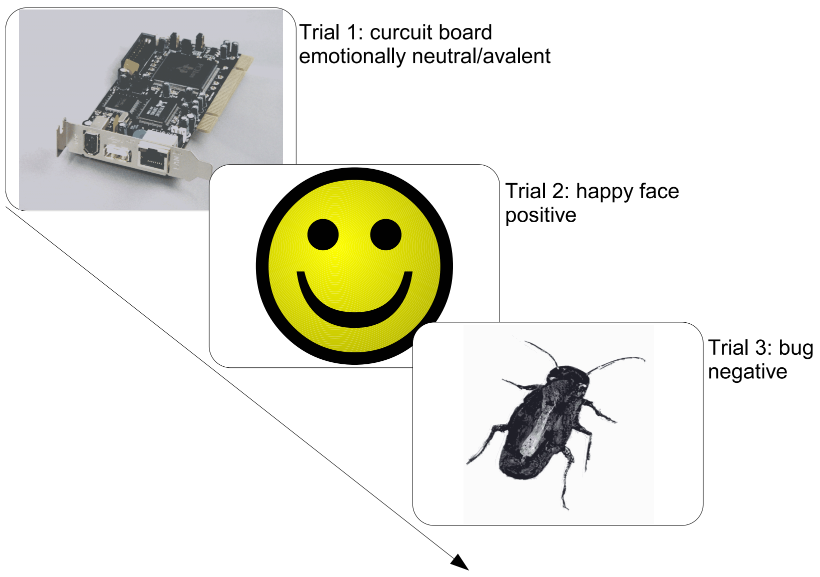


Figure 2.3: Emotional slides are shown in a random series.

In this first example, pictures of varying affective quality are presented to the subjects. The pictures belong to three different classes: positive, neutral, and negative.

The Wizard guides you through the settings. The navigation bar at the top always shows where you are; in the area on the right you may enter the data.

Firstly, specify a name for the design. This name is needed to identify the design. It will be used to name all files which RQUBƎ generates.

Since a negative picture may have different effects after a positive than after a negative picture, the transition frequencies are set tightly: first-order probability $Q_1 = 0.02$. Additionally, the distribution of pairs of consecutive pictures is set to a low value $Q_{D1} = 0.10$, in order to obtain a homogeneous distribution.

2.3.2 Flanker-Compatibility-Task

This second section introduces a larger design with two factors. It will explain the difference between a restrictions that are applied to the whole design, and those applied to a single factor. This example may be found at `examples\Flanker.qini` in your RQUBE installation directory.

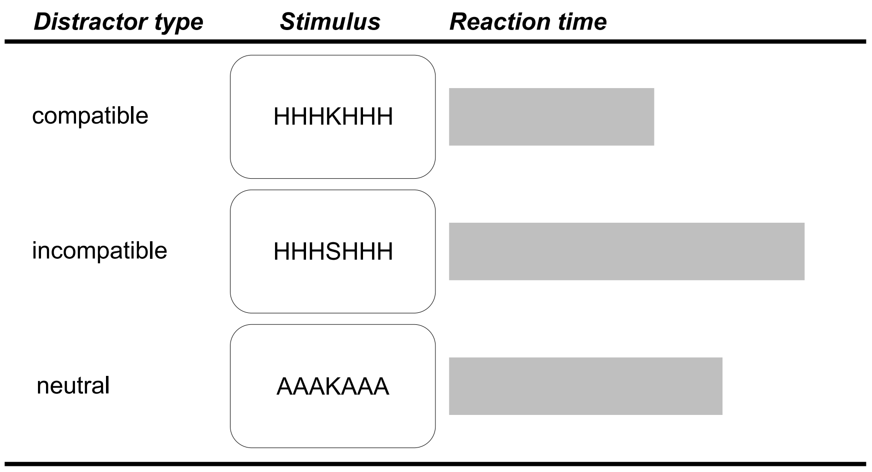


Figure 2.4: The letters H and K are assigned to the right hand response. The letter S is assigned to a left hand response. The letter A does never appear as target stimulus and is therefore not associated with any response.

A simple bimanual choice reaction task was chosen for these purposes. The well known flanker-task was invented by Eriksen and Eriksen (1974). One target stimulus is presented in each trial that is flanked by distractor stimuli. According to the quality of these distractors there are three experimental conditions. The distractors are assigned to a response that is opposite to the appropriate response; that means, the same distractor stimuli are used as target in other trial. This condition is called incompatible (sometimes incongruent). The second condition shows distractors, that require the same response as the target (again that means, the same distractor stimuli are used as target in other trial). This condition is called compatible (congruent, respectively). Thirdly, there is a control condition with distractors, that are not associated with any response. These stimuli are not part of the task set. This condition is called neutral.

If we now want to set up an flanker-experiment, we need one factor *distractor type* with three levels: *compatible*, *neutral*, and *incompatible*. Furthermore, we need to specify a factor *response hand* with two levels: *right* and *left*. This is necessary, because the type of reaction varies independently from the experimental distractor conditions. Since Bertelson (1965) found out that repeated responses have faster response times, responses have to be randomised. This second factor *response hand* can be added and edited just like the first one.

Now, since we have more then one factor, we have to differentiate between the restrictions set for each factor and those of the whole design. The factors comprise two and three levels, respectively. The designs number of levels is $2 \cdot 3 = 6$. The restrictions of each factor are only applied to its levels. The sequence is checked for as if only this one factor existed, any other factor is ignored. Therefore the design restrictions control the order of the combinations of factor levels.

2.3.3 Nogo-Task

Unfortunately, this feature has not been implemented, yet. Look out for the version 1.00 of RQUBƎ.

This third example introduces the feature of constraints. It is one weakness of RQUBƎ that all factor levels occur with equal frequencies. This weakness (among others) may be overcome by using constraints, which are exemplified in this section. This example may be found at `examples\Nogo.qini` in your RQUBƎ installation directory.

Using constraints it is possible to treat several levels of a factor as if they were equal. In Nogo-tasks there is a pre-potent response tendency that has to be in-

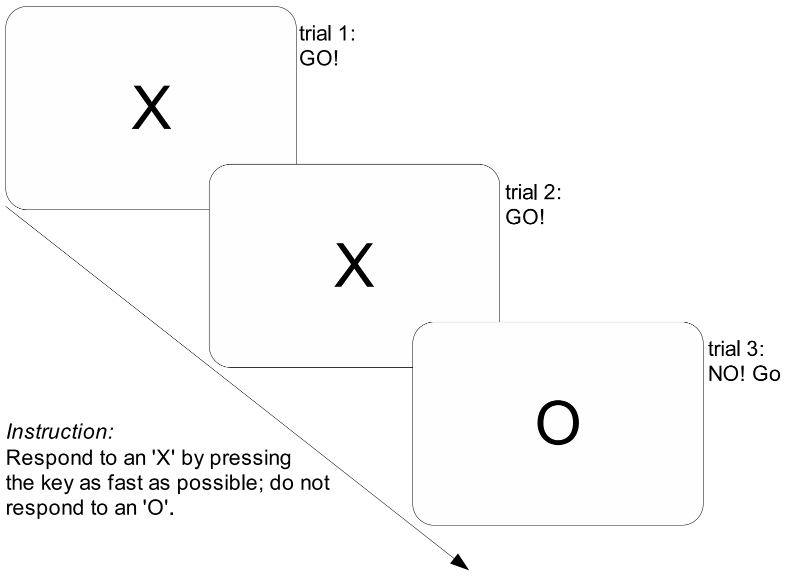

trial 1:
GO!
X
trial 2:
GO!
X
trial 3:
NO! Go
O
Instruction:
Respond to an 'X' by pressing the key as fast as possible; do not respond to an 'O'.

Figure 2.5: The NoGo-task: The letters X and O are presented in a random series. Subjects are instructed to respond to the X and to omit it, when the O appears.

hibited in several "Nogo-trials". Hence, it is common to have more Go– than Nogo-trials, in order to strengthen the response tendency. The properties of factors, however, do not allow unequal frequencies. This may be circumvented by using a level constraint. You define a factor with three levels. One level represents the Nogo-trials (*nogo*) and the other two are Go-trials (*go1* and *go2*). Then you define a constraint that treats *go1* and *go2* as if they were equal. For example, if the maximum run length (Q_{RL}) of the level constraint is set to 3, any series with more than 3 *go*-trials is invalid, irrespective of the fact that *go1*– and *go2*-trials are intermixed.

Table 2.2: Only three successive Go-trials are valid.

valid	go2, go1, go2, nogo, go2, go2, nogo
invalid	go2, go2, go2, **go1**, nogo, go1, go2

A level constraint is defined by a number of sets and the sequence restrictions (Q) that are applied to those sets. The elements in a set are the combinations of factor levels (cells) that have to be treated equally.

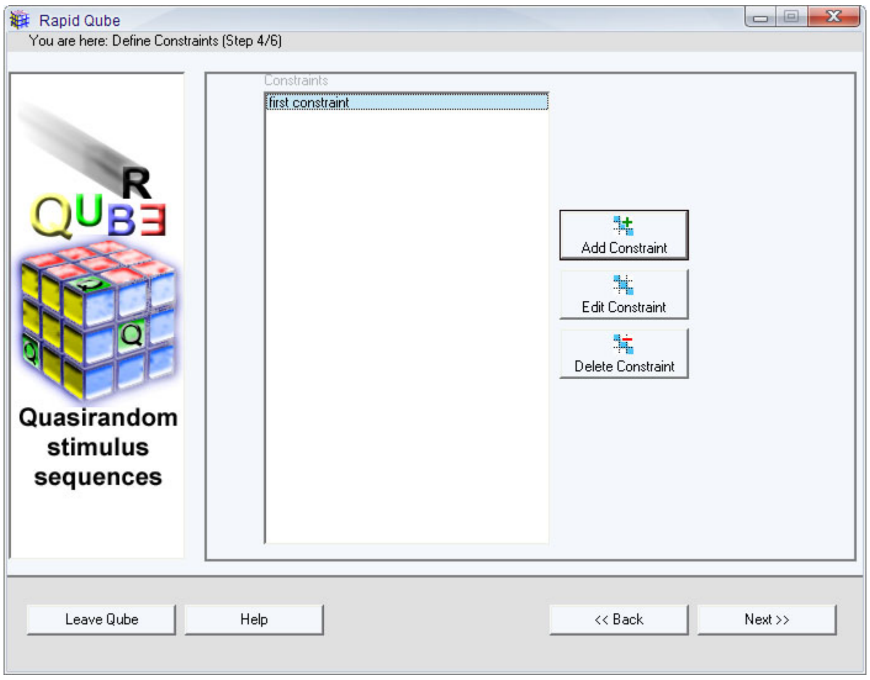


Figure 2.6: The fourth wizard page displays the constraints of the design.

You may add constraints on the fourth screen of the wizard (see figure 2.6). A click on the button "Add Constraint" leads to the constraint editor (see figure 2.7). The editor displays all combinations of factor levels (cells) of the design. You may combine any set of cells. To achieve this, click on the according cells in the left list and provide a name for the set in the input field "Name of Set". To complete this step click the "»" button. A new set will appear in the right list.

There is one aspect that you have to keep in mind when you set restrictions for level constraints. If different sets have different item frequencies several restrictions become more difficult to handle. If one set occurs twice as frequent as another set, it is quite obvious that the number of occurrences cannot be equal. Furthermore, it makes sense to set the maximum run length Q_{RL} to the a value that is at twice as big as the corresponding restriction of the Go-Nogo factor, because there are twice as many items within one constraint set than within each level of the Go-Nogo factor. Finally, unequal set frequencies affect all restrictions that are based on transition frequencies (the probability of first order Q_1 and the

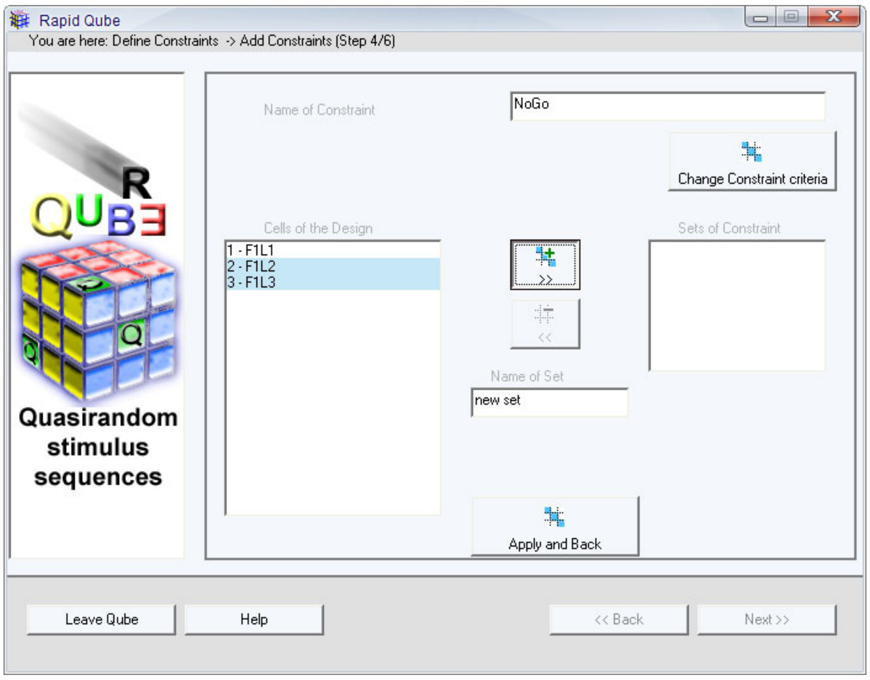


Figure 2.7: The constraint editor.

first-order distribution Q_{D1}; also: autocorrelation, runs test). In this example we simply switch the first-order restrictions off.

2.3.4 Negative Priming

We added the negative priming paradigm to this manual, in order to show that a solution in RQUBE is not always obvious, but – nevertheless – possible. We consider the negative location priming here (for an overview of this paradigm, see Fox, 1995; May, Kane, & Hasher, 1995). This example may be found at `examples\NegativePriming.qini` in your RQUBE installation directory.

In an negative location priming experiment each trial consists of two kinds of stimuli. One is the target that informs the subjects about the required reaction, the other is a distractor stimulus and has to be ignored. These two stimuli are distributed among (at least) four possible locations on the screen.

The negative priming effect occurs, when the target appears at the same location, the distractor was at the trial before; the distractor appears at a previously empty location (ignored repetition, IP). In the control condition both stimuli appear at previously empty locations (see figure 2.8).

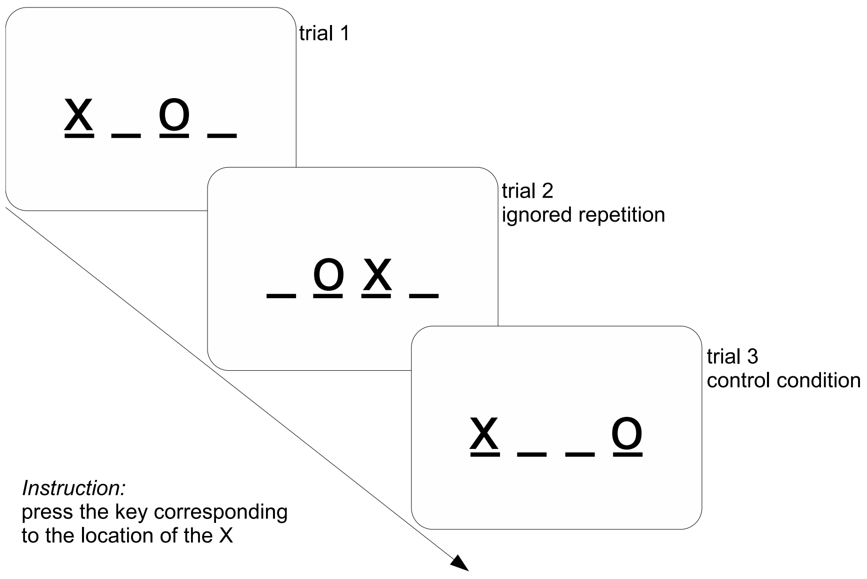


Figure 2.8:

When we set this experiment up, it took us some time to find a solution. Initially, we thought about a two-factor solution. One factor (4 levels) designates the location of the target and the second (3 levels) the location of the distractor. Unfortunately this does not work. The solution of this problem is not obvious.

The problem can be solved by a two factor RQUBƎ-design. The first factor defines, if the current trial is an ignored repetition or control condition. When it is ignored repetition, the location of the current target is clear. The experiment software simply needs to choose the location of the distractor in the previous trial. In the control condition two locations are available. Therefore, the second factor defines the position of the distractor stimulus, which may be more left or more right in the array. It not known, yet, which of the four locations the distractor will appear at. Again the experiment software has to analyse the previous trial and to select the two empty locations. The first will be assigned to the distractor, according to the value of the second RQUBƎ-factor, and the other location is left for the target. Now there is one problem left. The trial sequence does say nothing about the first stimulus. The experiment software has to choose a first stimulus array, maybe by random, maybe in a systematic way (e.g depending on the subject number).

2.3.5 Mismatch negativity paradigm

The mismatch negativity paradigm was chosen to exemplify the control for habituation effects with RQUBƎ. This example may be found at `examples\ MMN.qini` in your RQUBƎ installation directory.

The design chosen for this example is taken from Näätänen, Pakarinen, Rinne, and Takegata (2004). The mismatch negativity (MMN) is an electrocortical deflection, which occurs in response to a change in auditory stimulation. The experiment described by Näätänen et al. (2004) involves nine different stimuli, one frequent standard stimulus and eight different deviants (of 5 different classes: pitch, duration, intensity, sound-source location, interrupted).

The MMN after deviant stimuli is highly sensitive to stimulus repetitions, therefore standards and deviants are presented alternatingly. In Näätänen et al. (2004) blocks of 10 trials had been defined, which comprised 5 standards and 5 deviants. Since RQUBƎ cannot accomplish randomisation in exactly the same way, we chose the following procedure. Each trial had two sub-trials, the first one presented a standard stimulus and the second one of the deviants. Doing so, we merely had to randomise the deviant stimuli. This can be accomplished by a single factor "deviant" with eight levels: de–/increased pitch, de–/ increased duration, de–/ increased intensity, sound-source location, interrupted.

A "minimum distance" of 2 was selected, to ensure that deviants are not repeated. In doing so, there were at least three other stimuli between two equal deviants. In order to guarantee a good distribution of all distractors over the whole experiment the distribution of zero order (Q_{D0}) has been set to a relatively low level $Q_{D0} = 0.02$.

2.4 How to implement RQube output in E-Prime and Presentation

E-Prime

The files created by RQUBƎ can easily be implemented in E-Prime experiments. In RQUBƎ you need to choose the “E-Prime compatible” output format.

In E-Prime you just need to copy the following code into an inline-section, that is located shortly before the list, that should the trials. You have to replace `TrialList` by the name of your own list, `PATH` by the path of the list and `STEM` by the leading part of the random lists file name. If you need more information please refer to your E-Prime documentation.

```
TrialList.Filename = "PATH\\" + "STEM" +
                     c.GetAttrib("Subject") + ".qrnd"
TrialList.LoadMethod = ebLoadMethodFile
TrialList.Load
TrialList.Reset
```

Presentation

RQUBƎ has the ability to print Presentation SDL templates. In RQUBƎ you need to choose the “Presentation SDL” output format.

Raw output

Besides the E-Prime and Presentation support, RQUBƎ may save a simple text file with a number in each line that represents one combination of factor levels.

2.5 The logfile

There is no need to omit a sequence that does not match certain characteristics. The numerical values of these characteristics may be printed into a log file. The log file is a tab delimited ascii table. It can easily imported into standard statistic software (like SPSS, Statistica, SAS, MINITAB, or R).

2.6 How to generate (pseudo–) random numbers

In case the restrictions are tightly set the quality of the RNG is of minor interest. The built-in RNG in C++ may be selected, because of its computation speed, even though its performance is poor (Press, Teukolsky, Vetterling, & Flannery, 1992). If circumstances demand, to loosen the restrictions, the choice of the RNG may become a crucial issue. Therefore, RQUBƎ provides several methods to generate random numbers.

Three of four available methods are pseudo-RNGs, which means that they compute "random" numbers using a recursive algorithm. These RNGs need a seed value to start their computations. Equal seeds lead to equal series of random numbers. Usually in C++ the time is taken as seed. RQUBƎ provides improved seeding by combining several data (the current time, the position of the mouse pointer when the latest system message has been processed, etc.)

2.6.1 Built-in RNG

The RNG that is implemented in Borland C++ Builder(TM) is a pseudo random number generator of the form:

$$X_{j+1} = (aX_j + c) \mod m \tag{2.1}$$

With X_0 being any seed. This form is called the linear congruential method, and it is not really a good implementation of it (see Press et al., 1992). Since the

Table 2.3: Time consumption of the available random number generators. The values are only approximate. The first column shows the time needed to generate a single random number. The second column shows how many numbers are generated in one second. These (approximate) values were obtained on a 3GHz Pentium computer. The third column shows a machine independent estimate of time consumption. The 1.00 was assigned to the C++ RNG, the values of the other RNGs are determined relative to the C++ RNG.

Generator Type	*Time/Number (ms)*	*Numbers/Second*	*Relative Time*
Builtin C++	0.000299	3848	1.00
Mersenne Twister	0.000140	8790	0.47
Crypto-API	0.005015	217	16.76
Beecrypt Wave-In	38.187997	.03	127650.04

Mersenne Twister (section 2.6.2) has better random properties and is even faster, the built-in RNG should be second quality.

2.6.2 Mersenne Twister

The Mersenne Twister is a generator of pseudo-random numbers with excellent properties. Its qualities do not only include far better random numbers than the built-in C++ RNG, it is also faster. According to the authors, the Mersenne Twister provides a period of $2^{19937} - 1$ and equidistribution up to 32-Bit accuracy (Matsumoto & Nishimura, 1998).

2.6.3 Microsoft Cryptography API

Please refer to the description at the MSDN Library

2.6.4 "True" random numbers from Random.org

RQUBE uses the online service provided by random.org. Random.org is an online true random number generation service. The true random numbers are generated using atmospheric noise.

Because this RNG depends on an online service, the speed of the random number class depends on the speed of the internet connection.

This RNG depends on an existing internet connection. If you have a firewall, you must grant RQUBƎ access to random.org. Furthermore it needs an Internet Explorer 3.0 or later. If you have both and the RNG is still not working, contact your admin or computer centre.

2.6.5 "True" random numbers: Beecrypt Wave-in RNG

The Beecrypt is taken from the cryptography library Beecrypt. It is a random number generator that reads noise on the sound card microphone port. Hence, it does provide "true" instead of pseudo random numbers. If you intend to use it, be aware that this RNG is extremely slow. And please remove any recording devices from your microphone port.

The performance of this RNG depends on a number of factors, including processing power, sound card quality (cheap sound cards will likely produce more noise), sound card capabilities (sampling rate, sampling depth, number of channels) among others.

Important: We cannot deliver beecrypt with RQUBƎ, because they do not share the same license. If you want to use it, you need to get the library beecrypt.dll and store it in the program directory of RQUBƎ.

Chapter 3

Frequently asked questions (FAQ)

? I'm not completely happy with my output!

If any information is missing in the output files, you may always use a "find-and-replace" tool to modify them. There are several freeware tools can replace a specified text by another in more than just one file, e.g. Emurasofts "Replace in Files" (http://www.emurasoft.com/replall/index.htm) or the "Handy File Find and Replace" tool (http://silveragesoftware.com/handytools.html).

? How do I install RQUBƎ in Windows?

You need Windows 95/98/ME/NT4/2000/XP: Windows 3.11+win32s will not work. Your file system must allow long file names (as is likely except perhaps for some network-mounted systems). Make sure that you have the proper rights. They you simply may start the setup program, the wizard will guide you through the available options.

? How do I UNinstall RQUBƎ?

Normally you can do this from the RQUBƎ group on the Start Menu or from the Add/Remove Programs in the Control Panel. However – be honest, you do not really want to do that?

? Citing RQUBƎ

If you use **RQUBƎ** in a study and this study is going to be published, you are obliged to cite **RQUBƎ** as follows: In german publications: ? ((?)). RQube (Version 1.03.56). Universität Trier; abgerufen am xx.xx.200x unter: http://rqube.seifseit.de/

In english publications: Seifert, J. & Britz. P. (2007). RQube (Version 1.03.56). Universität Trier; retrieved xx.xx.200x, from http://rqube.seifseit.de/

Please do not forget to replace download date and version number.

? Does **RQUBƎ** use the Registry?

Yes. The instruction files (*.qini) are associated with the main program.

? What is the Registry?

The windows registry is a large database. Windows stores a lot of information in that database that it needs to run properly. E.g. the registry "knows" the program that windows shall use to open files from a certain type. If a file type shows an incorrect icon or opens a wrong program, there is a wrong association set in the registry. **RQUBƎ** associated the instruction files (*.qini) with the main program. Double clicking on an instruction file starts RQube.

? **RQUBƎ** does not skip the current sequence, even though the timeout has been reached

RQUBƎ computes the complete sequence at first. During computation (only the restrictions Q_{RL} and Q_0 are considered). The timeout is checked after the whole sequence is complete. If it has been reached, **RQUBƎ** skips the current sequence and tries the next one; if not, the remaining restrictions are checked for.

Chapter 4

Mathematical Background of the characteristics

At first we have to introduce some variables:

- C: is the complete number of combinations of factor levels realised in the design. In other words this is the number of cells in the experimental design.
- K: the combination of factor levels of interest.
- n_K: the number of occurrences of K in a trial sequence.
- L: the total number of elements of a trial sequence.
- d_i: the distance between the i-th and the $(i+1)$-th occurrence of K, except d_0 being the distance between the last and the first element.
- f_{ij}: observed frequency of an item-pair (i, j).
- $\hat{f}_{ij}$: expected frequency of an item-pair (i, j).

  Example:

$$l, m, l, l, K, \underbrace{K, m, K}_{d_2=2}, \ldots m, m, K, l$$

- D: is the mean distance between two elements.

Furthermore the length of the sequence is given by

$$L = \sum_{i=0}^{n_K} d_i \tag{4.1}$$

And the mean distance between two elements is

$$D = \frac{1}{n_K} \cdot \sum_{i=0}^{n_K} d_i = \frac{L}{n_K} \tag{4.2}$$

4.1 Zero order probability

Probability of first order (Q_1) simply means, that each item occurs with equal frequencies. Therefore RQUBE has to ensure that the number of occurrences exactly match $\frac{L}{C}$. If you require unequal frequencies you have to pool several treatment levels (using constraints) and treat them as if it was one. See section 2.3.3 for an example.

4.2 First order probability

Given the assumption that all item pairs are equiprobable $\hat{f}_{ij}$ is defined as follows:

$$\hat{f}_{ij} = \frac{L-1}{n_K^2} \tag{4.3}$$

The χ^2 is known as:

$$\chi^2 = \sum \frac{(f-\hat{f})^2}{\hat{f}} \tag{4.4}$$

This results in:

$$Q_1 = p(\chi^2_{df}) \quad \text{with } df = n_K^2 - 1 \tag{4.5}$$

4.3 Distribution of zero-order probability

Q_{D0} describes, how homogeneously the items are distributed in the trial sequence. What we need for this purpose is an index, that varies between 0 and 1 and changes with the distribution of the items. Q_{D0} can be described as squared deviation of the inter-item-distances from the optimal mean distances. This squared deviation is standardized by its minimum and maximum possible value.

Q_{D0} is defined as the standardized squared deviation of the distances between the occurrences of K. The squared deviation:

$$\sum_{i=1}^{n_K}(d_i - D)^2 \tag{4.6}$$

$$\Leftrightarrow \sum_{i=0}^{n_K}(d_i^2 - 2d_iD + D^2)$$

$$\Leftrightarrow \sum_{i=0}^{n_K} d_i^2 - \sum_{i=1}^{n_K} 2d_iD + \sum_{i=1}^{n_K} D^2$$

$$\Leftrightarrow \sum_{i=0}^{n_K} d_i^2 - 2D\sum_{i=1}^{n_K} d_i + n_KD^2$$

because of (2) follows:

$$\Leftrightarrow \sum_{i=0}^{n_K} d_i^2 - 2D \cdot Dn_K + n_KD^2$$

$$\Leftrightarrow \sum_{i=0}^{n_K} d_i^2 - n_KD^2 \tag{4.7}$$

The maximum possible value of equation (4.7) is:

$$Q_{D0}^{max} = (n_K - 1) + (L - n_K + 1)^2 - L^2/n_K \tag{4.8}$$

From (4.7) and (4.8) follows:

$$\mathcal{Q}_{D\prime} = \frac{\sum_{i=1}^{n_K} d_i^2 - n_KD^2}{(n_K - 1) + (L - n_K + 1)^2 - L^2/n_K} \tag{4.9}$$

In the following we will show, how the Q_{D0}^{max} is derived. The problem can be described geometrically (see figure 4.3). The d_i form a vector $\vec{d} = (d_1, d_2, ..., d_{n_K})$ in an n_K-dimensional space. Since all d_i are dependent from each other ($d_i = L - \sum_{j\neq i} d_j$) all possible vectors $\vec{d}$ form a plane E with $n_K - 1$ dimension. E can be written as:

$$E : \vec{d} = \vec{b}_1 + [\lambda_2 n_K(\vec{b}_2 - \vec{b}_1)] + \ldots + [\lambda_{n_K} n_K(\vec{b}_{n_K} - \vec{b}_1)] \tag{4.10}$$

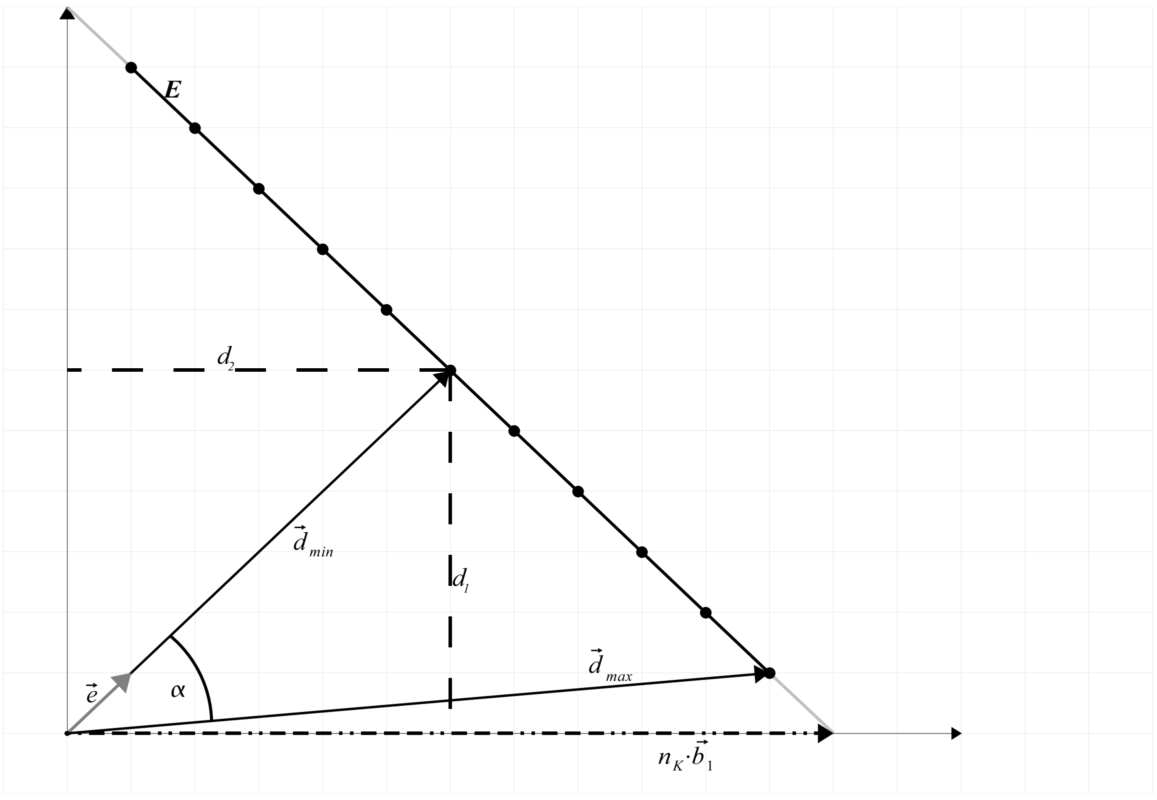


Figure 4.1: Illustration of the geometrical interpretation of Q_{D0} for the 2-dimensional case. The dots mark the possible points in the plane E, since all d_i are integer variables.

with b_i being the basis vectors and λ_i being an arbitrary value.

The vector $\vec{e} = (1, 1, \ldots, 1)$ can be shown to be orthogonal to E ($\vec{e} \perp E$). Any of the complanar vectors $\vec{b}_i - \vec{b}_1$ is orthogonal the $\vec{e}$:

$$\begin{aligned} \cos(\phi) &= \frac{\vec{e} \cdot n_K[\vec{b}_i - \vec{b}_1]}{|\vec{e}||n_K[\vec{b}_i - \vec{b}_1]|} = \frac{-n_K + n_K}{\sqrt{n_K}\sqrt{2n_K^2}} = 0 \\ &\Rightarrow \phi = 90^\circ \end{aligned} \tag{4.11}$$

The angle α can be calculated by

$$\sin\alpha = \frac{\vec{d}\vec{e}}{|\vec{d}||\vec{e}|} \tag{4.12}$$

Therefore the length of the vector $\vec{d}$ is

$$\begin{aligned}|\vec{d}| &= \frac{\vec{d}\vec{e}}{\cos\alpha|\vec{e}|} \\ &= \frac{\sum_i(d_i e_i)}{\cos\alpha|\vec{e}|} \qquad (e_i = 1) \\ &= \frac{\sum_i(d_i)}{\cos\alpha|\vec{e}|} \end{aligned} \tag{4.13}$$

The length of a vector $\vec{d}$ can also be written as follows

$$|\vec{d}| = \sqrt{\sum_{i=1}^{n_K} d_i^2} \tag{4.14}$$

When we combine the formulae (4.7), (4.14) and (4.13) the squared deviations can also be calculated by

$$\begin{aligned}\sum_{i=1}^{n_K}(d_i - D)^2 &= \sum_{i=1}^{n_K} d_i^2 - n_K D^2 = |\vec{d}|^2 - n_K D^2 \\ &= \left[\frac{\sum_i(d_i)}{\cos\alpha|\vec{e}|}\right]^2 - n_K D^2 \end{aligned} \tag{4.15}$$

We know that $\sum_i(d_i)$ is the length L of our trial sequence which is constant. Furthermore $|\vec{e}|$ is constant ($|\vec{e}| = \sqrt{n_K}$). So the length of $\vec{d}$ in equation (4.13) depends only on the $\sin\alpha$. $|\vec{d}|$ becomes greater with $\sin\alpha$ becoming smaller. And because $-45° \leq \alpha \leq +45°$ it can be concluded that the sum of the squared deviations becomes smallest, when its vector is orthogonal to E. And the sum of the squared deviations becomes greater with the deviation of α from $0°$. Its maximum is reached at $\pm 45°$. Furthermore the term $-n_K D^2$ is also a constant. This leads to the conclusion that $\sum_{i=1}^{n_K}(d_i - D)^2$ reaches its minimum with $\vec{d} \perp E$ or $\vec{d}||\vec{e}$ (i.e. $\vec{d} = const. \cdot \vec{e}$). In other words, the sum of squared deviations is minimal, when all d_i are equal. The more the d_i differ from each other, the bigger the angle α becomes and so does the sum of squared deviations. It reaches its maximum when all d_i equal to one except of one that equals to $L - n_K + 1$.

From this it follows that:

$$\begin{aligned}\sum_{i=0}^{n_K} d_i^2 - n_K D^2 &= \sum_{i=1}^{n_K} 1^2 + (L - n_K + 1)^2 - n_K D^2 \\ &= (n_K - 1) + (L - n_K + 1)^2 - n_K D^2 \\ &= (n_K - 1) + (L - n_K + 1)^2 - L^2/n_K \qquad (4.16)\end{aligned}$$

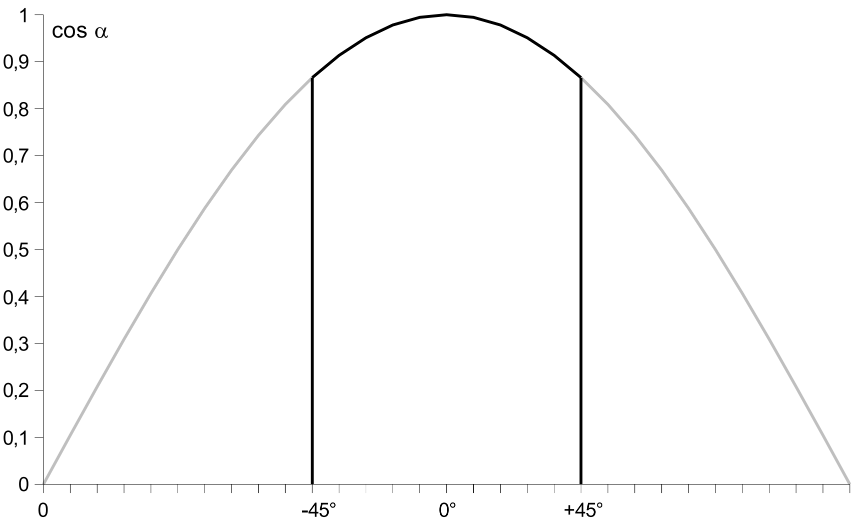


Figure 4.2: The cosine function peaks at a value of 0: $\cos(0) = 1$.

4.4 Distribution of first-order probability

The logic of the first-order distribution is equal to the zero-order distribution (see previous section).

4.5 Runs test

The mathematical background of the runs test is found in Mood (1940).

4.6 Autocorrelation

The auto-correlation is computed as follows:

$$r_l = \frac{\sum_{i=1}^{n_K - l} (x_i - \bar{x})(x_{i+l} - \bar{x})}{Var(x)} \tag{4.17}$$

with l being the current lag. Q_{auto} is defined as:

$$Q_{auto} = \max_{l=1}^{6}(r_l) \tag{4.18}$$

4.7 Standardized entropy

The entropy is defined as follows (Shannon, 1948):

$$H = \sum P(x) \cdot \log_2[P(x)] \tag{4.19}$$

To receive a characteristic with a fixed range of values, the entropy is divided by the maximum possible entropy.

$$Q_{info} = \max_{l=1}^{2} \frac{H_l}{H_{l,max}} \tag{4.20}$$

References

Bar-Hillel, M., & Wagenaar, W. A. (1991). The preception of randomness. *Advances in applied mathematics*, *12*, 428-454. 1.1.1, 1.2.3

Bertelson, P. (1965). Serial choice reaction-time as a function of response versus signal-and-response repetition. *Nature*, *206*(980), 217-218. 2.3.2

Chaitin, G. J. (1975). Randomness and mathematical proof. *Scientific American*, *232*(5), 47–52. 1.1.1

Emerson, P. L., & Tobias, R. D. (1995). Comuter program for quasi-random stimulus sequences with equal transition frequencies. *Behavior Research Methods, Instrumtents and Computers*, *27*(1), 88–98. 1.2.1, 4

Eriksen, B. A., & Eriksen, C. W. (1974). Effects of noise letters upon the identification of a target letter in a nonsearch task. *Perception and Psychophysics*, *16*(1), 143-149. 2.3.2

Falk, R., & Konold, C. (1998). Subjective randomness. In S. Kotz, C. B. Read, & D. L. Banks (Eds.), *Encyclopedia of statistical sciences* (p. 653-659). New York: J. Wiley & Sons. 1.1.1

Fox, E. (1995). Negative priming from ignored distractors in visual selection: A review. *Psychonomic Bulletin & Review*, *5*(2), 145-173. 2.3.4

Gammermann, A., & Vovk, V. (1999). Kolmogorov complexity: Sources, Theory and Applications. *The Computer Journal*, *42*(4), 252-255. 1.1.1

Gilmore, J. B. (1989). Randomness and the search for psi. *Journal of Parapsychology*, *53*(4), 309-340. 1.1

Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modelling and Computer Simulation*, *8*(1), 3-30. 2.6.2

May, C. P., Kane, M. J., & Hasher, L. (1995). Determinants of negative priming. *Psychological Bulletin*, *118*(1), 35-54. 2.3.4

Mood, A. (1940). The distribution theory of runs. *The Annals of Mathematical Statistics*, *11*, 367-392. 1.2.3, 4.5

Näätänen, R., Pakarinen, S., Rinne, T., & Takegata, R. (2004). The mismatch negativity (MMN): towards the optimal paradigm. *Clinical Neurophysiology*, *115*(1), 140-144. 2.3.5

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical recipes in C: The art of scientific computing* (2 ed.). Cambridge: Cambridge University Press. 2.6, 2.6.1

Seifert, J., & Britz, P. (in prep.). Qube or a simple, reliable solution for quasi-random stimulus sequences.

Shannon, C. E. (1948, July, October). A mathematical theory of communication. *The Bell System Technical Journal*, *27*, 379-423, 623-656. (Reprint) 4.7

Appendix A

Default values of restrictions

Table A.1: *Default values of the restrictions in Qube.*

	Symbol	Default status	Default value ($\check{Q}$)	Min.	Max.	Optimal value (Q^*)
Maximum run length	Q_{RL}	on	$L - 1/C^2$	1	C	–
Minimum distance	Q_{MD}	off	1	1	C	–
Zero-order probability	Q_0	on	L/C	L/C	L/C	0
First-order probability	Q_1	on	0.2	*0*	1	0
Distribution of zero order	Q_{D0}	on	0.30	*0*	1	0
Distribution of first order	Q_{D1}	off	0.30	*0*	1	0
Autocorrelation	Q_{auto}	off	0.20	*0*	1	0
Runs test	Q_{runs}	off	–	0	1	0.5
Standardized entropy	Q_H	off	0.8	0	*1*	1

Appendix B

Commented instruction file

Below is an example listed that explains the structure of RQUBƎ's instruction files, though we do not recommended that anyone edits these files. The RQUBƎ instruction files are simple text files without any formatting options. Any formatting here is only done to improve readability. Comments are in printed Times, the instructions are in Typewriter style. RQUBƎ treats any strings case-insensitive.

`; This file was generated by RQube`	
`; please do not edit`	only when you know, what you are doing
`[GENERAL]`	
`NUMBEROFTRIALS=`*`450`*	
`EXPNAME=`*`ERIKSONSREVENGE`*	points to design section (see below)
`HOWMANY=`*`20`*	sequence files from 0 to 19 will be
`TARGETPATH=`	where to write those files
`TIMEOUT=`	how long should computation of a single list be tried
`TRYOUT=`	how often should computation be tried
`RNG=`	Random number generator (0 = Built-in; 1 = Mersenne Twister; 2 = Microsoft Crypto-API; 3 = BeeCrypt)

OUTPUTSTYLE=	0 = raw data; 1 = E-Prime compatible; 2 = Presentation SDL-template
[ERIKSONSREVENGE]	**General design named Erikson's Revenge**
FACTORCOUNT=*3*	
FACTOR01=*DISTRACTOR*	Factors and their names
FACTOR02=*RESPONSEHAND*	each has its own section below
CONSTRAINTCOUNT=*1*	
CONSTRAINT01=*NONSENSE*	Level constraints and their names

The criteria below are computed on the basis of combinations of factor levels.
Any identifier of a RQube-restriction may occur here. If it does not,
the restriction is not applied here.

Q0=*5*	Maximum run length
MINDIST=*1*	Minimum distance
Q1=*75*	0. order probability
Q2=	1. order probability
Q3=	distribution of 0. order
Q4=	distribution of 1. order
[DISTRACTOR]	**Factor**: Distractor
NUMBEROFLEVELS=*3*	Number of levels
LEVELLABEL01=*incompatible*	Factor levels and their identifiers
LEVELLABEL02=*undefined*	
LEVELLABEL03=*compatible*	

The criteria below are computed on the basis of factor levels.
Any identifier of a RQube-restriction may occur here.
If it does not, the restriction is not applied to this factor.

Q0=*5*	
MINDIST=*1*	
Q1=*150*	= NUMBEROFTRIALS/NUMBEROFLEVELS Do not edit any other value here.
Q2=	
Q3=	

`Q4=`	
`[RESPONSEHAND]`	**Factor**: response hand
`NUMBEROFLEVELS=`*2*	.
`LEVELLABEL01=`*R*	.
`LEVELLABEL02=`*L*	.
`Q0=`*5*	
`Q1=`*225*	= `NumberOfTrials/NumberOfLevels` Do not edit any other value here.
`Q2=`	
`Q3=`	
`Q4=`	
`[NONSENSE]`	A level constraint; it does not make any sense here, it is just an example.
`NUMBEROFSETS=`*4*	
`SETLABEL01=`*S1*	
`SETLABEL02=`*S2*	
`SETLABEL03=`*S3*	
`SETLABEL04=`*S4*	
`SET01=`*0,1*	the indices of the cells in each set (comma separated)
`SET02=`*2,3*	
`SET03=`*4*	
`SET04=`*5*	
`Q0=`*6*	
`Q1=`*75*	equals Q_1 of the factor space
`Q2=`*-0.10*	negative value: Q_2 has been switched off
`RUNS=`*0.15*	
`[LOGFILE]`	
`Q0=`*0*	1 = print on; 0 = print off
`Q1=`*0*	
`Q2=`*1*	

`Q3=`*`1`*	
`Q4=`*`1`*	
`AUTOCORR=`*`1`*	
`RUNS=`*`1`*	
`ENTROPY=`*`1`*	

Appendix C

Error messages

The list below shows the error codes and messages fed back by the kernel of RQUBƎ. Usually, you should not see one of those errors, since the GUI has to preent these kinds of errors before they occur.

0	No errors
Internal errors	
0xC401	Invalid session handle
0xC402	No trials left in this design
0xC403	Number of trials does not fit to the design
0xC404	This error code is not used by Qube
0xC405	Cannot open additional Qube session
0xC406	Cannot initialize random number generator (RNG)
Output errors	
0xD001	Cannot open/ create file
Initialization errors	
0xC801	Invalid number of levels (too few or too many)
0xC802	Invalid number of factors (too few or too many)
0xC803	Invalid number of trials (too few or too many)
0xC804	Number of cells exceeds maximum

0xC805	Invalid number of sets (too few or too many)
0xC806	No parameter file specified
0xC807	Number of levels does not fit numerb of trials
0xC808	Constraint has an invalid set definition
0xC80A	Restriction “Q1” is not valid
0xC80B	“Q1” cannot be disabled
0xC80C	The selected random number generator (RNG) cannot be initialized. Try another one.
Unspecified errors	
0xD801	An unspecified error has occurred
0xD802	Unspecified error - Exit is the only option
0xD803	Unspecified error when closing kernel
0xD804	Unspecified error when computing sequence
0xD805	Unspecified error when starting computation
0xD806	Unspecified error when reading parameter file
0xD807	Unspecified error when initializing kernel

List of Corrections