To appear in *Advances in Physics*
Vol. 00, No. 00, Month 20XX, 1–17

PAPER

A computational violation of CHSH with a local model.

Han Geurdes[a*]

[a]*C. vd Lijnstraat 164, 2593 NN Den Haag Netherlands*

(*v0.1 released July 2015*)

In this paper the design and coding of a local hidden variables model is presented that violates the $|CHSH| \leq 2$ criterion in size larger than $1 + \sqrt{2}$.

PACS: 03.65.Ud Entanglement, 02.60.Cb Numerical simulation

Keywords: computer simulation, CHSH violation with local model, entanglement.

1. Introduction.

The CHSH inequality is an important element in the discussion about the existence or nonexistence of additional local hidden parameters [1]. The CHSH inequality [2] is derived from Bells formula for the correlation [3], $E(a, b)$, between distant spin measurements with setting parameters a and b. Generally,

$$E(a, b) = \int d\lambda \rho_\lambda A_\lambda(a) B_\lambda(b) \tag{1}$$

In (1) we can identify the probability density $\rho_\lambda \geq 0$, with $\int d\lambda \rho_\lambda = 1$. The λ are introduced to explain the correlation and need to have a local effect. This can e.g. be accomplished [5] if a λ_1 is assigned to the A measurement instrument and λ_2 to the B instrument. Furthermore, the measurement functions $A_\lambda(a)$ and $B_\lambda(b)$ both project in $\{-1, 1\}$ to represent binairy spin variables (e.g. up=1, down=-1). The CHSH inequality is based on the following expression,

$$S = E(1, 1) - E(1, 2) - E(2, 1) - E(2, 2) \tag{2}$$

The quartet of setting pairs $\mathcal{Q} = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$ occurs random in a series of N spin measurements of entangled particle pairs. Alice and Bob are two assitents in the experiment who, per trial or particle pair measurement, randomly select the setting of their measurement instrument. The argument in favor of the CHSH inequality [4] and against a possible probability loophole [5] is as follows. From (1) and (2) we may write, suppressing the hidden variables index λ, notation for the moment,

$$S = E\{A(1)[B(1) - B(2)] - A(2)[B(1) + B(2)]\}. \tag{3}$$

*Corresponding author. Email: han.geurdes@gmail.com

According to [4], because, A and B are both $\in \{-1,1\}$, we see that when $B(1)=B(2)$, then $S=\pm 2$, while, when $B(1)=-B(2)$, it again flollows, $S=\pm 2$. Hence, $|S|$ based on (1) cannot be larger than 2 and therefore the nonzero probability of $|S|>2$ with a local hidden variables model of [5] must be based on a mistake. It will be demonstrated with a numeric simulation using local principles that this claim is untrue.

2. Reformulation Bells formula.

Let us define basic sets derived from the difference $E(a,b)-E(x,y)$, (a,b) and (x,y) are different settings. We have, $\Omega_+(a,b;x,y)=\{\lambda\,|\,A_\lambda(a)B_\lambda(b)=A_\lambda(x)B_\lambda(y)=+1\}$, together with $\Omega_-(a,b;x,y)=\{\lambda\,|\,A_\lambda(a)B_\lambda(b)=A_\lambda(x)B_\lambda(y)=-1\}$ and $\Omega_0(a,b;x,y)=\{\lambda\,|\,A_\lambda(a)B_\lambda(b)=-A_\lambda(x)B_\lambda(y)=\pm 1\}$. The three sets are disjoint and if Λ denotes the universe set of the λ variables we also have $\Lambda=\Omega_+(a,b;x,y)\cup\Omega_-(a,b;x,y)\cup\Omega_0(a,b;x,y)$. Note that in $E(a,b)-E(x,y)$ only the $\lambda\in\Omega_0(a,b;x,y)$ contribute. Therefore

$$E(a,b)-E(x,y)=-2\int_{\lambda\in\Omega_0(a,b;x,y)}A_\lambda(x)B_\lambda(y)d\lambda \tag{4}$$

If subsequently, $E(a,b)=0$ and we write $\Omega'_0(x,y)=\Omega_0(a,b;x,y)$ and (a,b) such that $E(a,b)=0$, then

$$E(x,y)=2\int_{\lambda\in\Omega'_0(x,y)}\rho_\lambda A_\lambda(x)B_\lambda(y)d\lambda \tag{5}$$

With $E(x,y)=E_T(x,y)$. Subsequently from $E(a,b)=0$ it follows [5] that,

$$E_C(x,y)=2\int_{\lambda\in\Omega'_+(x,y)}\rho_\lambda d\lambda-2\int_{\lambda\in\Omega'_-(x,y)}\rho_\lambda d\lambda \tag{6}$$

and, of course, $E_C(x,y)=E(x,y)$ via $E(a,b)=0$. Like in [5] we take the probability density, $\rho_\lambda=\rho_{\lambda_1,\lambda_2}$ and $\rho_{\lambda_1,\lambda_2}=\rho_{\lambda_1}\rho_{\lambda_2}$. The separate λ_1, is assigned to A and λ_2, is assigned to B. For, $j=1,2$,

$$\rho_{\lambda_j}=\begin{cases}\frac{1}{\sqrt{2}}, & \lambda_j\in\left[-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right]=\Lambda_j\\ 0, & \lambda_j\notin\left[-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right]\end{cases} \tag{7}$$

with the universal set, $\Lambda=\Lambda_1\times\Lambda_2$. Furthermore, $\Omega'_\pm(x,y)$, is the Cartesian product of a λ_1 and a λ_2 interval, i.e. $\Omega'_\pm(x,y)=\Omega'_{A\pm}(x)\times\Omega'_{B\pm}(y)$. Similar as in [5] let us take

$$\begin{array}{l}\Omega'_{A\pm}(1)\in\left\{\emptyset,\{\lambda_1\,|\,-1+\frac{1}{\sqrt{2}}\le\lambda_1\le\frac{1}{\sqrt{2}}\}\right\},\ \Omega'_{B\pm}(1)\in\left\{\emptyset,\{\lambda_2\,|\,-\frac{1}{\sqrt{2}}\le\lambda_2\le 0\}\right\}\\ \Omega'_{A\pm}(2)\in\left\{\emptyset,\{\lambda_1\,|\,-\frac{1}{\sqrt{2}}\le\lambda_1\le 1-\frac{1}{\sqrt{2}}\}\right\},\ \Omega'_{B\pm}(2)\in\left\{\emptyset,\{\lambda_2\,|\,0<\lambda_2\le\frac{1}{\sqrt{2}}\}\right\}\end{array} \tag{8}$$

The measurement functions to be used in the numerical simulations are

$$A_{\lambda_1}(x)=\begin{cases}\alpha(x), & \lambda_1\in I(x)\\ \mathrm{sgn}\{\zeta(x)-\lambda_1\}, & \lambda_1\in\Lambda_1\backslash I(x)\end{cases} \tag{9}$$

Table 1. Initial first step combinations of N-dimensional a and$b$ arrays

first N pairs	second N pairs	third N pairs	fourth N pairs
(1,1)	(1,2)	(2,1)	(2,2)
(1,2)	(1,1)	(2,2)	(2,1)
(2,1)	(2,2)	(1,1)	(1,2)
(2,2)	(2,1)	(1,2)	(1,1)

With $I(x) \subset \Lambda_1$ and $\zeta(x) \in \Lambda_1 \backslash I(x)$. For $B_{\lambda_2}(y)$ a similar form is given, $\beta(y) = \pm 1$,

$$B_{\lambda_2}(y) = \{ \begin{array}{c} \beta(y), \qquad\qquad \lambda_2 \in J(y) \\ \mathrm{sgn}\{\eta(y) - \lambda_2\}, \lambda_2 \in \Lambda_2 \backslash J(y) \end{array} \tag{10}$$

3. Design of the numerical local model

The violation with the use of a computer model is performed in two steps. In the first step the A and$B$ of a local model are generated with a numerical expression of the mathematics presented in the previous two sections. This is done in a orderly fashion for setting parameters a and b. We select for Alice $1_A = (1, 0, 0)$ and $2_A = (0, 0, 1)$. For Bob we select $1_B = (-0.1568, 0.3482, -0.9242)$ and $2_B = (-0.8492, 0.06045, 0.5246)$. The selection for B parameters is approximative. It represents a check if quantum mechanical correlation values are reproduced. In the second step the a and b setting arrays are randomized. We take $N = 1 \times 10^4$ pairs per setting combination. So a typical randomized a array will look like $(1, 2, 1, 1, 2, ..., 1)$ and holds $4N$ entries. Similarly for the randomized b array. Each, CHSH quartet combination (a, b) with $a \in \{1, 2\}$ and $b \in \{1, 2\}$ will hold exactly N times a replication of $(1, 1)$ "measurements", N times $(1, 2)$, N times $(2, 1)$ and N times $(2, 2)$.

3.1. *First step*

In the first step the following four possible matchings of ordered a and b arrays are represented in table - 3.1. The structure presented in equations (9) and (10) are implemented in the following snippet of (pseudo) R-code

$$\begin{array}{l} if(a == 1)\{ \\ \qquad if((\lambda_1 < 1/\sqrt{2})\ \&\&(\ \lambda_1 > -1 + (1/\sqrt{2}))\{ \\ \qquad\qquad A[a, \lambda_1] = \alpha[a] \\ \qquad \} \\ \qquad if((\lambda_1 > -1/\sqrt{2})\,\&\&(\lambda_1 < -1 + (1/\sqrt{2}))\{ \\ \qquad\qquad A[a, \lambda_1] = sign[\zeta[a] - \lambda_1] \\ \qquad \} \\ \} \end{array}$$

The λ_1 in the previous snippet of code runs between $-\frac{1}{\sqrt{2}}$ and $\frac{1}{\sqrt{2}}$ and is discretized with $d\lambda_1 = (1/\sqrt{2})/(N/2)$, i.e., for index i_A in 1 to$N$, stepsize 1, we have $\lambda_1 = -\frac{1}{\sqrt{2}} + i_A * d\lambda_1$.

For, $a = 2$ we have

$$
\begin{aligned}
&if(a == 2)\{\\
&\quad if((\lambda_1 < 1 - 1/\sqrt{2})\ \&\&(\ \lambda_1 > -(1/\sqrt{2}))\{\\
&\quad\quad A[a, \lambda_1] = \alpha[a]\\
&\quad\}\\
&\quad if((\lambda_1 > 1 - 1/\sqrt{2})\,\&\&(\lambda_1 < (1/\sqrt{2}))\{\\
&\quad\quad A[a, \lambda_1] = sign[\zeta[a] - \lambda_1]\\
&\quad\}\\
&\}
\end{aligned}
$$

The employed sign function is an abbreviation of the assignment of -1 and $+1$ in the code. The ζ parameter is: $\zeta[1] = -0.3127620$ and $\zeta[2] = 0.0150918$. The $\alpha[a]$ is in $\{-1, 1\}$ as in equation (9). The A results are stored on file and the two snippets of code stand in a loop which runs from $i_A = 1$ to N with stepsize 1 each time computing a new λ_1 for given a, α and ζ parameters. A similar procedure is performed for B. We have, similarly, $d\lambda_2 = (1/\sqrt{2})/(N/2)$ in a separate loop running over $i_B = 1$ to N, stepsize 1, $\lambda_2 = -\frac{1}{\sqrt{2}} + i_B * d\lambda_2$.

$$
\begin{aligned}
&if(b == 1)\{\\
&\quad if((\lambda_2 > 0\ \&\&(\ \lambda_2 < 1/\sqrt{2}))\{\\
&\quad\quad B[b, \lambda_2] = sign[\beta[b] - \lambda_2]\\
&\quad\}\,else\,\{\\
&\quad\quad B[b, \lambda_2] = \beta[b]\\
&\quad\}\\
&\}
\end{aligned}
$$

and for $b = 2$

$$
\begin{aligned}
&if(b == 2)\{\\
&\quad if((\lambda_2 < 0\ \&\&(\ \lambda_2 > -1/\sqrt{2}))\{\\
&\quad\quad B[b, \lambda_2] = sign[\beta[b] - \lambda_2]\\
&\quad\}\,else\,\{\\
&\quad\quad B[b, \lambda_2] = \beta[b]\\
&\quad\}\\
&\}
\end{aligned}
$$

The η parameters are, $\eta[1, j] = 0.2678 + (0.1608 * sign(0.5 - runif(1)))$ and $\eta[2, j] = -0.1880 - (0.1747 * sign(0.5 - runif(1)))$. With runif(1) a single random number is generated by R. The first index is the setting b. The second index is related to the selection of the series. We have for b in table - 3.1 either, alternating, $b = 1$, or, $b = 2$, i.e. the first and the third row in the table, or $b = 2$ and $b = 1$, i.e. the second and the fourth row. In order to make the j index independent from the selected a (i.e. which row in the table-3.1) we select the row type, i.e. begin with $a = 1$ and continue with $a = 2$, like in the first and second row of the table, or begin with $a = 2$ and continue with $a = 1$, like in the third and fourth row, randomly. So, B may have a random j variation in the η parameter but it doesn't convey anything about the a 's used by Alice. The employed code is $r = runif(1)$ and subsequently $if(r < 0.5)\{\gamma = 0\}else\{\gamma = 1\}$ and $if(\gamma == 0)\{aSet[1] = 1, aSet[2] = 2\}else\{aSet[1] = 2, aSet[2] = 1\}$. If $aSet[1] = 1$ we have rows 1 or 2 of the table-3.1. If $aSet[1] = 2$, we have rows 3 or 4 of that same table.

A similar structure for B exists. However, we select using a $\delta = 0$ or $\delta = 1$ parameter. E.g., $\delta = 0$ and $if(\delta == 0)\{bSet[1] = 1, bSet[2] = 2\}else\{bSet[1] = 2, bSet[2] = 1\}$ gives $bSet[1] = 1$ and $bSet[2] = 2$.

The runif(1) selection of γ hides the a information from B. The subsequent selection of $\delta = 0$ and $\delta = 1$ connects two B series to one a series. We note that A and B are determined independently, i.e. locally, and that a information is hidden from B's use of the j index of $\eta[b, j]$. Hence we may claim that both A and B series are determined based on local parametrization. If the i_A and i_B loops are run and i_B run for $\delta = 0$ and for $\delta = 1$ we have one A series and two B series independent of each other's information. Both $\delta = 0$ and $\delta = 1$ series have probability nonzero to generate CHSH violations of size $\geq 1 + \sqrt{2}$. The violating A and $B_{\delta=0}$ together with $B_{\delta=1}$ are stored on external separate files named EPRA, EPRB0 and EPRB1. The employed settings are also written to external files. For details the reader is referred to appendix A.

3.2. *Second step*

After clearing the memory in R, the three files are, with a second program, read into arrays, A, $B0$ and $B1$. The a and b arrays are also read and are used in a randomization of a simulation of an experiment using a and b arrays. We note that a information is hidden from B and, vice versa, b information is hidden from A. The randomization of the a array is done with a large $4N$ sided dice. A side of that dice contains, in addition to either, $a = 1$ or $a = 2$, the ranking number of a in the original sequence. So, suppose we had, $\gamma < 0.5$ in the first step, then the first $2N$ entries of the a array are 1 and the remaining $2N$ entries are 2. The first entry $a = 1$ has ranking number 1, the second $a = 1$ has 2 etcetera. The $2N$-th $a = 1$ will have ranking number $2N$. Subsequently, the first $a = 2$ will have $2N + 1$ etc.

For the source in the experiment we introduce the array S. If, for example, the $2N + 1$ array entry $a = 2$ is randomly selected as the first one after throwing of the A dice, then $s = 2N + 1$ will be loaded as the first entry into the source array, i.e. $S[1] = 2N + 1$. The activity of source in the simulation is represented by an array S and the first value of this array is $s = 2N + 1$. This value will be send to B for the first pair in the simulation.

Note that a information is hidden from B and that doesn't change when an entry is moved around in the a array. It also doesn't change when its ranking number is send from the source to B in the actual simulation. B is unaware of a. I.e. we are allowed to load a source array S with ranking numbers of a entries and send it to B because B cannot know the value of a when γ is selected randomly. The ranking number can be seen as a kind of "random seed" for all elements in the experiment. When the seed changes, Bob only knows that there will be a change in all elements using that same seed. When B does not know a, the specific numerical value remains hidden despite sending ranking information of a to B. A change in ranking number in the arays is for B, a change from $a = 1$ to the next $a = 1$ or from $a = 1$ to the next $a = 2$ or from $a = 2$ to $a = 1$ or from $a = 2$ to $a = 2$. The hiding of information via $\gamma \sim runif(1)$ and the selection rule in the first setp on the side of A only, is crucial to warrant the locality of the information.

In the program the arrays $aAliceWork$ and $bBobWork$ are randomly construed with the use of a dice. The original ordered a and b arrays (see table - 3.1) in the first step are loaded and transformed into $aAliceWork$ and $bBobWork$. The sequences for b from the first step are read from file and kept in $bBob0$ and $bBob1$. If $nTest = 4N$, the crucial

part of the second step program is given by

$$
\begin{array}{l}
AOut < -array(0, c(2, nTest)) \\
BOut < -array(0, c(2, nTest)) \\
for(i\, in\, seq(1, nTest))\{ \\
\qquad a = aAliceWork[i] \\
\qquad b = bBobWork[i] \\
\qquad s = S[i] \\
\qquad AOut[a, i] = A[s] \\
\qquad if(bBob[s, 1] == b)\{ \\
\qquad\quad BOut[b, i] = B0[s] \\
\qquad \}else\{ \\
\qquad\quad BOut[b, i] = B1[s] \\
\qquad \} \\
\}
\end{array}
$$

So if the randomly selected b from $bBobWork$ at trial number i is equal to the s-th element in $bBob0$, represented by, $bBob[s, 1]$, then the simulated B response, $BOut[b, i]$ is equal to $B0[s]$, i.e. the under $\delta = 0$ generated B from the first step. If $bBob[s, 1] == b$ is not true, then b is in $bBob1$, or $bBob[b, 2]$. The B simulated measurement is then $B1[s]$, i.e. the B generated in the first step under $\delta = 1$. The $AOut[a, i]$ and $BOut[b, i]$ arrays are the simulated "measurements" in the computer program.

In the second program we check

$$CHSH = EOut[2, 2] - EOut[1, 2] - EOut[2, 1] - EOut[1, 1].$$

The $EOut$ is determined with $EOut < -array(0, c(2, 2))\, for(a\, in 1 : 2)\{\, for(b\, in 1 : 2)\{\, EOut[a, b] = sum(AOut[a,] * BOut[b,])/N\,\}\}$. It gives repeatedly in one run $CHSH \approx 2.437$ with roughly a ratio $CHSH/(1 + \sqrt{2}) \approx 1.01$. With "min" operation searches in the $AOut$ and $BOut$ arrays we established that no $A = 0$ or $B = 0$ "measurements" occur. For details the reader is referred to appendix B.

4. Conclusion

In the paper a numerical simulation is given for local violation of the CHSH criterion. The mathematics of [5] is used and can be found in the defense [6] against the points raised by Gill in [4].

In the present simulation paper, use is made of the fact that four basic configurations in a and b arrays size $N = 1 \times 10^4$ per setting combination, can be violated with the use of the mathematics of [5]. The basic configurations consist of e.g. first $N \times (1, 1)$ pairs, then $N \times (1, 2)$ pairs , then $N \times (2, 1)$ pairs and finally $N \times (2, 2)$ pairs. Similar basic configurations can be foud in the table of the body of the present paper. Using a random selection $\gamma \sim runif(1)$ the A side $4N$, a array of Alice plus the $4N$ array carrying the $A \in \{-1, 1\}$, can be created and hidden from Bob. Bob, in turn, may generate two B sequences. If Alice has via $\gamma \sim runif(1)$ the series of $2N \times 1$ first entries in the a array and then $2N \times 2$ in the a array, then Bob may store in the first preparatory step, two violating B sequences, one B sequence, the $\delta = 0$, is $N \times 1$, then, $N \times 2$, then $N \times 1$ and finally $N \times 2$. Or the $\delta = 1$ series, $N \times 2$, then, $N \times 1$, then $N \times 2$ and finally $N \times 1$. The B computation outcome is stored in B_δ.

The simulation step, i.e. the second step of the program is in fact a completely new program that starts from a cleared environment (in R). The in the first step stored a series is randomized in $aAliceWork$ and the rank numbers of the randomized a are stored in an array S representing the source. Bob randomizes his b setting array similarly. Alice doesn't make use of the ranking numbers of Bobs array. The first steps in this program are preparatory. It is used to detach the generating principle in the first step from the simulation in the second step.

Subsequently, a selection rule over $4N$ trials, with a in $aAliceWork$ and b in $bBobWork$ settings using A and B_δ gives a one run violation of the CHSH. The B instrument contains B_0 and B_1 and actualizes in $BOut$ via the source S array containing ranking information.

The sharing of ranking number of $aAliceWork$ with Bob via the source array does not give away the hidden information of the numerical values in the $aAliceWork$ array. This is so because the $\gamma \sim runif(1)$ selection in the first step on the A side, hides the A side from Bob. Alice has no knowledge of Bob's b selections in $bBobWork$ either. Note that in the second step the S array can be in a source server standing between two client computers representing A and B. So the three computers, A, S and B can completely mimic the experimental set-up if this is considered necessary.

The claim made in [4] is mathematiucally and numerically invalid. The violation is in an ideal situation where 100% detection efficiency exists. It therefore, like [5] and [6] questions the soundness of the principle of the CHSH and rejects [4] that " in [5] there must be an error" is based on biasedness towards the soundness of CHSH. Gill admits [7] that rejection of a probability loophole in the CHSH cannot be valid when a local numerical simulation, such as in the present paper, can be accomplished.

We conclude that nature can generate violations of the CHSH criterion with the use of local hidden variables. From the results of simulatiuon, entanglement of particles and/or measuring instruments appears to be a mutual sharing of a "random seed" where numerical information is encapsulated in the particular element that participates in the experiment.

References

[1] A. Einstein, B. Podolsky, N. Rosen, (1935), *Can quantum-mechanical description of physical reality be considered complete, Phys. Rev.* **47** 777-780.

[2] J.F. Clauser, M.A. Horne, A. Shimony, R.A. Holt, (1969), *Proposed experiment to test local hidden-variables theories, Phys. Rev. Lett.* **23** 880-884.

[3] J.S. Bell, (1964), *On the Einstein Podolsky Rosen paradox, Physics* **1** 195-200.

[4] R.D. Gill, (2015), *No probability loophole in the CHSH, Results in Physics* **5**, 156-157, http://dx.doi.org/10.106/j.rinp.2015.06.002.

[5] J.F. Geurdes, (2014), *A probability loophole in the CHSH, Results in Physics* **4**, 81-82, http://dx.doi.org/10.106/j.rinp.2014.06.002.

[6] J.F. Geurdes, (2015), *Why one can maintain there is a probability loophole in the CHSH*, viXra, 1507.0041v2.

[7] R.D. Gill (2015), *Personal communication.*

Appendix A. Code first step

Perhaps the reader must be warned that the two presented R programs here are not "optimally" programmed.

```
path<-"C:/Users/Han/Documents/R/"
outEPRA<-paste0(path,"EPRA.txt")
outEPRB0<-paste0(path,"EPRB0.txt")
outEPRB1<-paste0(path,"EPRB1.txt")
outaAlice<-paste0(path,"aAlice.txt")
outbBob0<-paste0(path,"bBob0.txt")
outbBob1<-paste0(path,"bBob1.txt")
#
DateTime<-date()
n=1e4
S=0
m<-0
Slim=1+sqrt(2)
#Slim=1.0
aAlice<-array(0,n)
bBob<-array(0,n)
while(abs(S)<Slim && m<100){
  m=m+1
  m1=m
  #(paste0(c("##"),DateTime), file=writeFile,append=TRUE)
  dlambda_1=(1/sqrt(2))/(n/2)
  dlambda_2=(1/sqrt(2))/(n/2)
  lambda_1<-array(0,n)
  zeta<-array(0,2)
  eta<-array(0,c(2,2))
  zeta[1]=-0.31276201747484
  zeta[2]=0.0150918216743739
  alpha<-array(0,2)
  alpha[1]=-1
  alpha[2]=alpha[1]
  beta<-array(0,2)
  beta[1]=-1
  beta[2]=beta[1]
  for (lambda_1_n in seq(1,n)){
    lambda_1[lambda_1_n]=(-1/sqrt(2))+(dlambda_1*lambda_1_n)
  }
  lambda_2<-array(0,n)
  for (lambda_2_n in seq(1,n)){
    lambda_2[lambda_2_n]=(-1/sqrt(2))+(dlambda_2*lambda_2_n)
  }
  aSet<-array(0,2)
  bSet<-array(0,2)
  r=runif(1)
  if(r<0.5){
    gamma=0
  }else{
    gamma=1
  }
  if(gamma==0){
    aSet[1]=1
```

```
  aSet[2]=2
}else{
  aSet[1]=2
  aSet[2]=1
}
delta=0
if(delta==0){
  bSet[1]=1
  bSet[2]=2
}else{
  bSet[1]=2
  bSet[2]=1
}
A<-array(0,c(2,n))
B<-array(0,c(2,n))
ATest<-array(0,4*n)
BTest<-array(0,4*n)
aAlice<-array(0,4*n)
bBob<-array(0,4*n)
E<-array(0,c(2,2))
m2=0
m3=0
m4=0
for( i in seq(1,2)){
  a=aSet[i]
    for (lambda_1_n in seq(1,n)){
      m3=m3+1
      if(a==1){
        if (lambda_1[lambda_1_n]<(1/sqrt(2)) && lambda_1[lambda_1_n]>-1+(1/sqrt(2))){
          A[a,lambda_1_n]=alpha[a]
          #ATest[m3]=A[a,lambda_1_n]
        }
        if (lambda_1[lambda_1_n]>-(1/sqrt(2)) && lambda_1[lambda_1_n]< -1+(1/sqrt(2)))
          if(lambda_1[lambda_1_n] < zeta[a]){
            A[a,lambda_1_n]=1
            #ATest[m3]=A[a,lambda_1_n]
          }else{
            A[a,lambda_1_n]=-1
            #ATest[m3]=A[a,lambda_1_n]
          }
        }
        if (lambda_1_n==n && A[a,n]==0){
          A[a,n]=1
          #ATest[m3]=A[a,lambda_1_n]
        }
      }
      if(a==2){
        if (lambda_1[lambda_1_n]<1-(1/sqrt(2)) && lambda_1[lambda_1_n]>-(1/sqrt(2))){
          A[a,lambda_1_n]=alpha[a]
          #ATest[m3]=A[a,lambda_1_n]
```

```
      }
      if (lambda_1[lambda_1_n]>1-(1/sqrt(2)) && lambda_1[lambda_1_n]< (1/sqrt(2))){
        if(lambda_1[lambda_1_n] < zeta[a]){
          A[a,lambda_1_n]=1
          #ATest[m3]=A[a,lambda_1_n]
        }else{
          A[a,lambda_1_n]=-1
          #ATest[m3]=A[a,lambda_1_n]
        }
      }
      if (lambda_1_n==n && A[a,n]==0){
        A[a,n]=-1
        #ATest[m3]=A[a,lambda_1_n]
      }
      A[a,lambda_1_n]<--A[a,lambda_1_n]
      #ATest[m3]=A[a,lambda_1_n]
    }
  }
#
for (j in seq(1,2)){
  b=bSet[j]
  #plot(lambda_1,A)
  dlambda_2=(1/sqrt(2))/(n/2)
  eta[1,j]=0.2678+(0.1608*sign(0.5 - runif(1)))
  eta[2,j]=-0.1880-(0.1747*sign(0.5 - runif(1)))
  for (lambda_2_n in seq(1,n)){
    m4=m4+1
    m2=m2+1
    if(b==1){
      if (lambda_2[lambda_2_n]>0 && lambda_2[lambda_2_n]<(1/sqrt(2))){
        B[b,lambda_2_n]=sign(eta[b,j]-lambda_2[lambda_2_n])
        BTest[m4]=B[b,lambda_2_n]
      }else{
        B[b,lambda_2_n]=beta[b]
        BTest[m4]=B[b,lambda_2_n]
      }
      if (lambda_2_n==n && B[b,n]==0){
        B[b,n]=-1
        BTest[m4]=B[b,lambda_2_n]
      }
    }
    if(b==2){
      if (lambda_2[lambda_2_n]<0 && lambda_2[lambda_2_n]>-(1/sqrt(2))){
        B[b,lambda_2_n]=sign(eta[b,j]-lambda_2[lambda_2_n])
        BTest[m4]=B[b,lambda_2_n]
      }else{
        B[b,lambda_2_n]=beta[b]
        BTest[m4]=B[b,lambda_2_n]
      }
      if (lambda_2_n==n && B[b,n]==0){
```

```
            B[b,n]=1
            BTest[m4]=B[b,lambda_2_n]
          }
          B[b,lambda_2_n]<--B[b,lambda_2_n]
          BTest[m4]=B[b,lambda_2_n]
        }
        aAlice[m2]=a
        bBob[m2]=b
      }

      #plot(lambda_2,B)
      E[a,b]=sum(A[a,]*B[b,])/n
    }#b
  }#a
  #print(E)
  S=-E[1,1]-E[1,2]-E[2,1]+E[2,2]
  if(abs(S)>2*sqrt(2)){
    S=0
    m=0
  }
}#S
if(gamma==0){
  ATest[1:n]=A[1,]
  xl=n+1
  xu=(2*n)
  ATest[xl:xu]=A[1,]
  xl=(2*n)+1
  xu=(3*n)
  ATest[xl:xu]=A[2,]
  xl=(3*n)+1
  xu=4*n
  ATest[xl:xu]=A[2,]
}else{
  ATest[1:n]=A[2,]
  xl=n+1
  xu=(2*n)
  ATest[xl:xu]=A[2,]
  xl=(2*n)+1
  xu=(3*n)
  ATest[xl:xu]=A[1,]
  xl=(3*n)+1
  xu=4*n
  ATest[xl:xu]=A[1,]
}
print(paste0("Number of pairs: ",n))
print(paste0("number of attempts to succes= ",m1))
print("====================== Internals ====================")
#print(paste0("zeta[1]=",zeta[1]))
#print(paste0("zeta[2]=",zeta[2]))
#print(c("Eta matrix: "))
```

```
print(eta)
print(E)
print(S)
#
write(BTest,file=outEPRB0,append=FALSE)
write(bBob,file=outbBob0,append=FALSE)
#
write(ATest,file=outEPRA,append=FALSE)
write(aAlice,file=outaAlice,append=FALSE)
##########################################################################################
DateTime<-date()
n=1e4
S=0
m<-0
Slim=1+sqrt(2)
#Slim=1.0
aAlice<-array(0,n)
bBob<-array(0,n)
while(abs(S)<Slim && m<100){
  m=m+1
  m1=m
  #(paste0(c("##"),DateTime), file=writeFile,append=TRUE)
  dlambda_1=(1/sqrt(2))/(n/2)
  lambda_1<-array(0,n)
  zeta<-array(0,2)
  eta<-array(0,c(2,2))
  zeta[1]=-0.31276201747484
  zeta[2]=0.0150918216743739
  alpha<-array(0,2)
  alpha[1]=-1
  alpha[2]=alpha[1]
  beta<-array(0,2)
  beta[1]=-1
  beta[2]=beta[1]
  for (lambda_1_n in seq(1,n)){
    lambda_1[lambda_1_n]=(-1/sqrt(2))+(dlambda_1*lambda_1_n)
  }
  lambda_2<-array(0,n)
  for (lambda_2_n in seq(1,n)){
    lambda_2[lambda_2_n]=(-1/sqrt(2))+(dlambda_2*lambda_2_n)
  }
  aSet<-array(0,2)
  bSet<-array(0,2)
#
  if(gamma==0){
    aSet[1]=1
    aSet[2]=2
  }else{
    aSet[1]=2
    aSet[2]=1
```

```
}
delta=1
if(delta==0){
  bSet[1]=1
  bSet[2]=2
}else{
  bSet[1]=2
  bSet[2]=1
}
A<-array(0,c(2,n))
B<-array(0,c(2,n))
ATest<-array(0,4*n)
BTest<-array(0,4*n)
aAlice<-array(0,4*n)
bBob<-array(0,4*n)
E<-array(0,c(2,2))
m2=0
m3=0
m4=0
for( i in seq(1,2)){
  a=aSet[i]
  for (lambda_1_n in seq(1,n)){
    m3=m3+1
    if(a==1){
      if (lambda_1[lambda_1_n]<(1/sqrt(2)) && lambda_1[lambda_1_n]>-1+(1/sqrt(2))){
        A[a,lambda_1_n]=alpha[a]
        #ATest[m3]=A[a,lambda_1_n]
      }
      if (lambda_1[lambda_1_n]>-(1/sqrt(2)) && lambda_1[lambda_1_n]< -1+(1/sqrt(2))){
        if(lambda_1[lambda_1_n] < zeta[a]){
          A[a,lambda_1_n]=1
          #ATest[m3]=A[a,lambda_1_n]
        }else{
          A[a,lambda_1_n]=-1
          #ATest[m3]=A[a,lambda_1_n]
        }
      }
      if (lambda_1_n==n && A[a,n]==0){
        A[a,n]=1
        #ATest[m3]=A[a,lambda_1_n]
      }
    }
    if(a==2){
      if (lambda_1[lambda_1_n]<1-(1/sqrt(2)) && lambda_1[lambda_1_n]>-(1/sqrt(2))){
        A[a,lambda_1_n]=alpha[a]
        #ATest[m3]=A[a,lambda_1_n]
      }
      if (lambda_1[lambda_1_n]>1-(1/sqrt(2)) && lambda_1[lambda_1_n]< (1/sqrt(2))){
        if(lambda_1[lambda_1_n] < zeta[a]){
          A[a,lambda_1_n]=1
```

```
      #ATest[m3]=A[a,lambda_1_n]
    }else{
      A[a,lambda_1_n]=-1
      #ATest[m3]=A[a,lambda_1_n]
    }
  }
  if (lambda_1_n==n && A[a,n]==0){
    A[a,n]=-1
    #ATest[m3]=A[a,lambda_1_n]
  }
  A[a,lambda_1_n]<--A[a,lambda_1_n]
  #ATest[m3]=A[a,lambda_1_n]
 }
}
#
for (j in seq(1,2)){
 b=bSet[j]
 #plot(lambda_1,A)
 eta[1,j]=0.2678+(0.1608*sign(0.5 - runif(1)))
 eta[2,j]=-0.1880-(0.1747*sign(0.5 - runif(1)))
 for (lambda_2_n in seq(1,n)){
  m4=m4+1
  m2=m2+1
  if(b==1){
    if (lambda_2[lambda_2_n]>0 && lambda_2[lambda_2_n]<(1/sqrt(2))){
      B[b,lambda_2_n]=sign(eta[b,j]-lambda_2[lambda_2_n])
      BTest[m4]=B[b,lambda_2_n]
    }else{
      B[b,lambda_2_n]=beta[b]
      BTest[m4]=B[b,lambda_2_n]
    }
    if (lambda_2_n==n && B[b,n]==0){
      B[b,n]=-1
      BTest[m4]=B[b,lambda_2_n]
    }
  }
  if(b==2){
    if (lambda_2[lambda_2_n]<0 && lambda_2[lambda_2_n]>-(1/sqrt(2))){
      B[b,lambda_2_n]=sign(eta[b,j]-lambda_2[lambda_2_n])
      BTest[m4]=B[b,lambda_2_n]
    }else{
      B[b,lambda_2_n]=beta[b]
      BTest[m4]=B[b,lambda_2_n]
    }
    if (lambda_2_n==n && B[b,n]==0){
      B[b,n]=1
      BTest[m4]=B[b,lambda_2_n]
    }
    B[b,lambda_2_n]<--B[b,lambda_2_n]
    BTest[m4]=B[b,lambda_2_n]
```

```
        }
        aAlice[m2]=a
        bBob[m2]=b
      }

      #plot(lambda_2,B)
      E[a,b]=sum(A[a,]*B[b,])/n
    }#b
  }#a
  #print(E)
  S=-E[1,1]-E[1,2]-E[2,1]+E[2,2]
  if(abs(S)>2*sqrt(2)){
    S=0
    m=0
  }
}#S
if(gamma==0){
  ATest[1:n]=A[1,]
  xl=n+1
  xu=(2*n)
  ATest[xl:xu]=A[1,]
  xl=(2*n)+1
  xu=(3*n)
  ATest[xl:xu]=A[2,]
  xl=(3*n)+1
  xu=4*n
  ATest[xl:xu]=A[2,]
}else{
  ATest[1:n]=A[2,]
  xl=n+1
  xu=(2*n)
  ATest[xl:xu]=A[2,]
  xl=(2*n)+1
  xu=(3*n)
  ATest[xl:xu]=A[1,]
  xl=(3*n)+1
  xu=4*n
  ATest[xl:xu]=A[1,]
}
print(paste0("Number of pairs: ",n))
print(paste0("number of attempts to succes= ",m1))
print("====================== Internals ====================")
#print(paste0("zeta[1]=",zeta[1]))
#print(paste0("zeta[2]=",zeta[2]))
#print(c("Eta matrix: "))
print(eta)
print(E)
print(S)
#
write(BTest,file=outEPRB1,append=FALSE)
```

```
write(bBob,file=outbBob1,append=FALSE)
```

Appendix B. Second step

```
path<-"C:/Users/Han/Documents/R/"
#
aAlice<-paste0(path,"aAlice.txt")
bBob0<-paste0(path,"bBob0.txt")
bBob1<-paste0(path,"bBob1.txt")
EPRA<-paste0(path,"EPRA.txt")
EPRB0<-paste0(path,"EPRB0.txt")
EPRB1<-paste0(path,"EPRB1.txt")
#
n=1e4
nTest=4*n
aAlice<-scan(aAlice)
aAliceWork<-aAlice
bBob<-array(0,c(nTest,2))
bBob[,1]<-scan(bBob0)
bBob[,2]<-scan(bBob1)
A<-scan(EPRA)
B0<-scan(EPRB0)
B1<-scan(EPRB1)
S<-array(0,nTest)
#a random
iTrial<-seq(1,nTest)
for(tel in 1:8){
  for(i in seq(1,nTest)){
    inDx<-as.integer(i+(nTest+1-i)*runif(1))
    iHelp<-iTrial[inDx]
    iTrial[inDx]<-iTrial[i]
    iTrial[i]<-iHelp
  }
}
for (i in seq(1,nTest)){
  iNdx<-iTrial[i]
  S[i]<-iNdx
  aAliceWork[i]<-aAlice[iNdx]
}
#b random
bBobWork<-array(0,nTest)
iTrial<-seq(1,nTest)
for(tel in 1:8){
  for(i in seq(1,nTest)){
    inDx<-as.integer(i+(nTest+1-i)*runif(1))
    iHelp<-iTrial[inDx]
    iTrial[inDx]<-iTrial[i]
    iTrial[i]<-iHelp
  }
```

```
}
for (i in seq(1,nTest)){
  iNdx<-iTrial[i]
  bBobWork[i]=bBob[iNdx,1]
}

AOut<-array(0,c(2,nTest))
BOut<-array(0,c(2,nTest))
#
for (i in seq(1,nTest)){
  a=aAliceWork[i]
  b=bBobWork[i]
  s=S[i]
  AOut[a,i]=A[s]
  if (bBob[s,1]==b){
    BOut[b,i]=B0[s]
  }else{
    BOut[b,i]=B1[s]
  }
}
EOut<-array(0,c(2,2))
for(a in 1:2){
  for (b in 1:2){
    EOut[a,b]=sum(AOut[a,]*BOut[b,])/n
  }
}
print(EOut)
SOut=EOut[2,2]-EOut[2,1]-EOut[1,2]-EOut[1,1]
print(paste0("CHSH=",SOut))
print(paste0("Ratio, Sout/(1+sqrt(2))=",SOut/(1+sqrt(2))))
x=EOut[1,1]
y=1-(EOut[1,1]^2)-(EOut[1,2]^2)
y=sqrt(y)
z=EOut[1,2]
o=EOut[2,1]
p=1-(EOut[2,1]^2)-(EOut[2,2]^2)
p=sqrt(p)
q=EOut[2,2]
bAr1<-array(0,3)
bAr2<-array(0,3)
bAr1[1]=x;bAr1[2]=y;bAr1[3]=z
bAr2[1]=o;bAr2[2]=p;bAr2[3]=q
len1=sqrt(t(bAr1)%*%bAr1)
len2=sqrt(t(bAr2)%*%bAr2)
print(" ")
print(paste0("setting Alice, 1=(1,0,0), 2=(0,0,1)"))
print(paste0("setting Bob 1=(",x,",",y,",",z," )"))
print(paste0("setting Bob 2=(",o,",",p,",",q," )"))
print(paste0("Length (setting Bob 1) = ",len1))
print(paste0("Length (setting Bob 2) = ",len2))
```