Shiny App - Simultaneous Goodness of Fit Testing

This web site describes how to run the simultaneous goodness-of-fit test described in . Because it was written with data from experiments in High Energy Physics and Astrophysics in mind it has some restrictions, but it should be useful in a number of other fields as well.

The app can be run online at https://drrolke.shinyapps.io/simgof/. If you have R/RStudio installed you can download the code from https://github.com/WolfgangRolke/simgof, unzip into the working directory and run it with

```
runApp()
```

The general setup and the features of the program are:

there are observations (events) $x_1, .., x_n$ and we wish to test whether $X_i \sim F$, F some probability distribution.

- n may be fixed or come from a Poisson distribution rate λ.
- $x_1, .., x_n$ may be continuous or binned.
- F may be fixed completely or the parameters can be estimated.

The null distributions are found via Monte Carlo simulation, so even when (say) the Kolmogorov-Smirnov statistic is used the fact that the distribution of the test statistic may be known is ignored. In this way we need not concern ourselves with questions such as under what conditions the large sample approximation is valid. The main advantage of using a limiting distribution is the speed of calculation. In most cases the routine will provide an answer in a matter of seconds, so this advantage is not crucial. Moreover, in most of the circumstances at least some of the test statistics do not have a known null distribution and so a Monte Carlo study is unavoidable at any rate.

When using the tool for several data sets and/or models it is highly recommended to get a "fresh" copy by reloading the tool first.

General Setup

The Data

The observations $x_1, .., x_n$ are either continuous observations or binned data such as one would draw in a histogram. In either case the data should be in an ASCII file that can be the uploaded to the web site. The formats are as follows:

a. Data continuous: file is a simple listing of the data, for example

0.738 0.951 0.49 0.586 0.628 0.635 0.794 0.125 0.795 0.458
0.881 0.49 0.506 0.62 0.602 0.944 0.796 0.83 0.62 0.999
0.164 0.568 0.757 0.978 0.658 0.738 0.947 0.663 0.757 0.985
0.356 0.517 0.677 0.663 0.985 0.787 0.664 0.525 0.742 0.544
...
0.474 0.592 0.998 0.85 0.689 0.538 0.894 0.919 0.868 0.849
0.525 0.693 0.606 0.428 0.778 0.195 0.728 0.783 0.449 0.483

b. Data binned: In the first column it should have the endpoints of the bins and in the second column the bin counts, for example:

0.00 23
0.01 25
0.02 31
...
0.99 10
1.00

Several examples are shown later.

Routines

For each case the following routines are needed:

a. *pnull* cumulative distribution function under the null hypothesis

b. *rnull* generate data under the null hypothesis

c. *qnull* quantile function under the null hypothesis. Only needed if ppcc or RGb tests are to be included.

d. *estimate* routine to estimate parameters from data. Only needed is parameters are to be estimated.

These routines should be saved as a single ASCII file that needs to be uploaded. Each of them can be written in either R or C++. R is generally recommended because it will be somewhat faster.

Included Examples

In order to allow users to become familiar with the program we will now discuss a number of examples. In each case the data has 1000 events.

Example 1: Uniform, Continuous data

We want to test whether a data set comes from a U[10,20] distribution. To create your own example open RStudio and run

```
write(round(runif(1000, 10, 20), 3), "dta.dat")
```

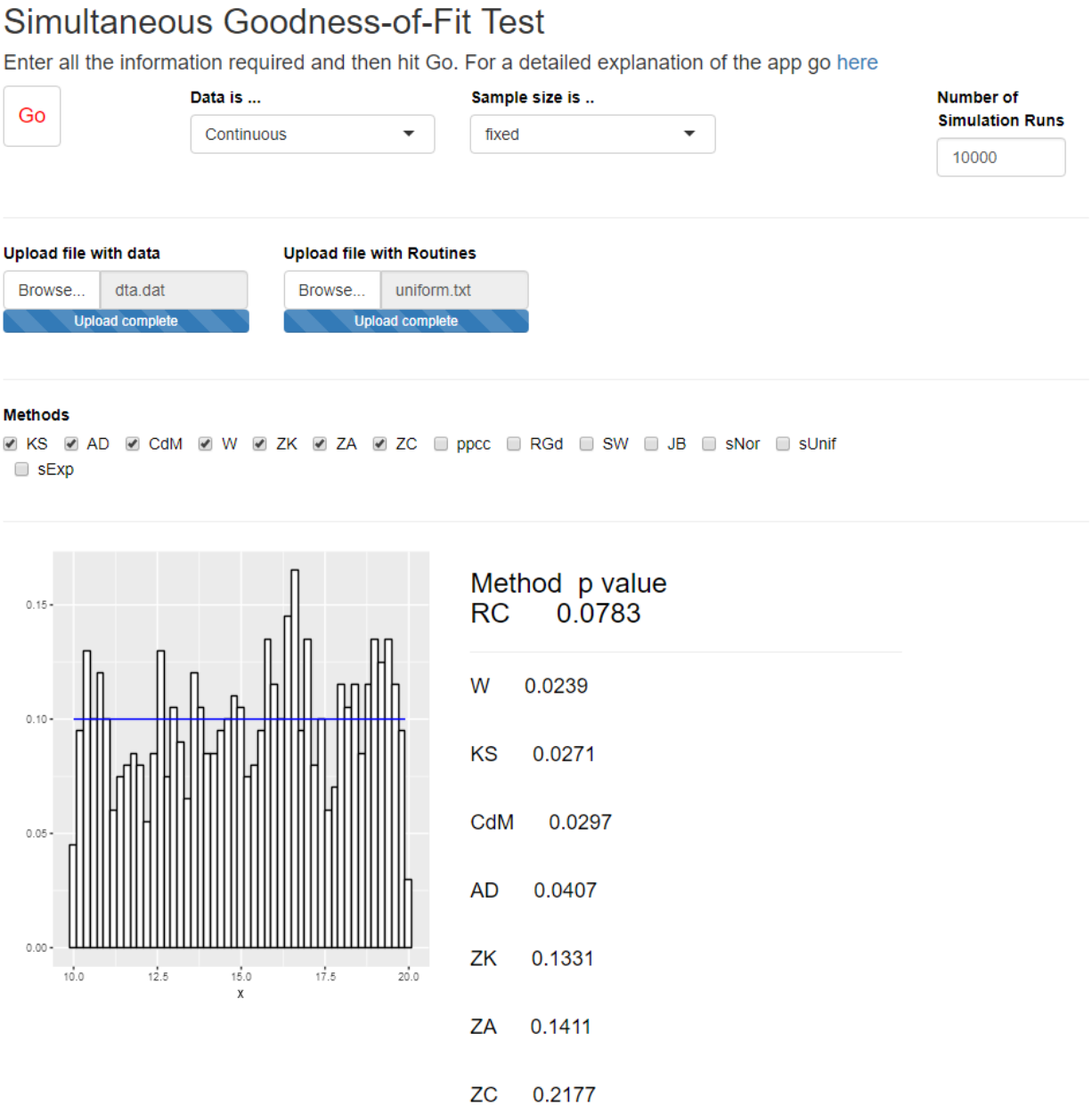
Now there is a file called dta.dat in the working directory of RStudio, which is at

```
getwd()
```

Next we need to create the file with the routines. To do so create an ASCII file, save it as *uniform.txt* (say) and copy paste the following code into it:

```
pnull <- function(x) {
  A=10
  B=20
  punif(x, A, B)
}
rnull <- function(n)  {
  A=10
  B=20
  runif(n, A, B)
}
```

Open the app, upload the files and hit Go. After a few seconds you should see this:



So the p value of the RC method is 0.0783.

Comments:

- The names of the routines have to be pnull, rnull, qnull and estimate
- If the word param appears anywhere in the code parameter estimation is done.
- Because the null hypothesis is uniform we could have included the sUnif test (Neyman's smooth test for uniform). We can still do this by checking the box and hitting Go again.
- Notice that any parameters (here A and B) are part of the routine, not arguments, unless we wish to fit for them.
- Here no parameter estimation is done, so there is no routine *estimate*.

Alternatively we could define the routines using C++. In that case the file uniform.txt will look like this:

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
NumericVector pnull(NumericVector x) {
  int n=x.length();
  double A=10;
  double B=20;
  NumericVector y(n);
  y=(x-A)/(B-A);
  return y;
}

#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
NumericVector rnull(int n) {
  NumericVector y(n);
  double A=10;
  double B=20;
  y=runif(n, A, B);
  return y;
}
```

Notice that the version of C++ used here is somewhat unusual: it allows vectorized arithmetic and the use of R code (y=runif(n, A, B)). For more details see http://rcpp.org/. However, someone not familiar does not need to use these features except for the header part and the special data types such as NumericVector.

Example: Normal with Parameter Estimation, Continuous Data

Here the null hypothesis spezifies the normal distribution and the mean and standard deviation are estimated via maximum likelihood.

Data:

```
write(round(rnorm(1000, 100, 10 ), 3), "shiny/dta.dat")
```

R routines:

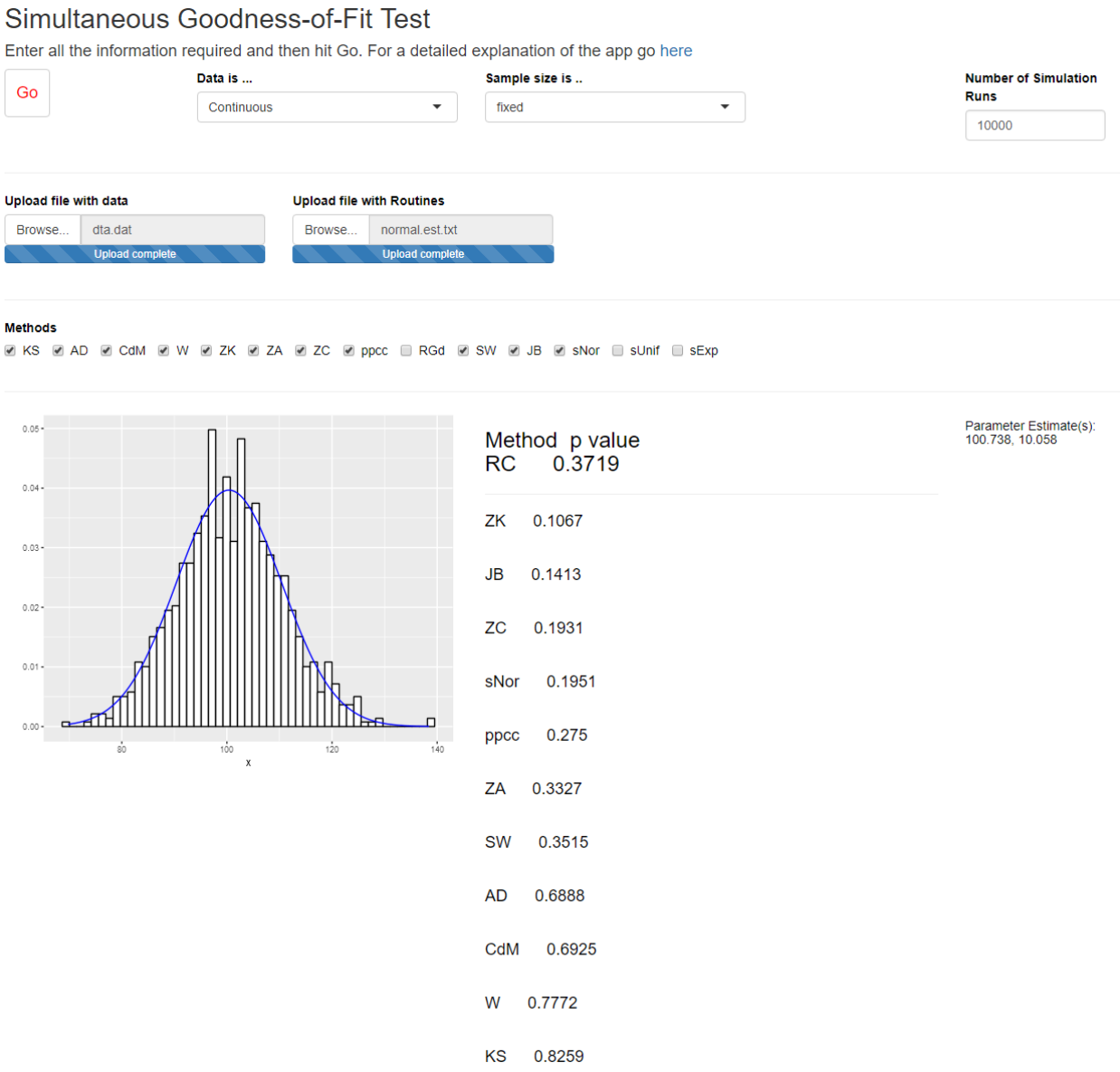
```
pnull <- function(x, param=c(0,1)) pnorm(x, param[1], param[2])
rnull <- function(n, param=c(0,1))  rnorm(n, param[1], param[2])
qnull <- function(x, param=c(0,1)) qnorm(x, param[1], param[2])
estimate <- function(x)  c(mean(x), sd(x))
```

C++ routines

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
NumericVector pnull(NumericVector x, NumericVector param) {
  int n=x.length();
  NumericVector y(n);
  y=pnorm(x, param[0], param[1]);
  return(y);
}

#include <Rcpp.h>
using namespace Rcpp;
```

```
// [[Rcpp::export]]
NumericVector rnull(int n, NumericVector param) {
  NumericVector y(n);
  y=rnorm(n, param[0], param[1]);
  return(y);
}

#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
NumericVector qnull(NumericVector x, NumericVector param) {
  int n=x.length();
  NumericVector y(n);
  y=qnorm(x, param[0], param[1]);
  return(y);
}

#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
NumericVector estimate(NumericVector x) {
  NumericVector y(2);
  y[0]=mean(x);
  y[1]=sd(x);
  return(y);
}
```

Beacuse the null hypothesis specifes the normal distribution we add the SW, ppcc, JB and sNor tests:



Example: Truncated Exponential

Data:

```
rnull <- function (n, param, A = 0, B = 1) {
    x <- NULL
    repeat {
        x <- c(x, rexp(n, param))
        x <- x[x > A]
        x <- x[x < B]
        if (length(x) > n)
            break
    }
    x[1:n]
}
x <- rnull(1000, 1/150, 100, 200)
```

```
write(x, "shiny/dta.dat")
```

```
## Error in file(file, ifelse(append, "a", "w")): cannot open the connection
```

R code:

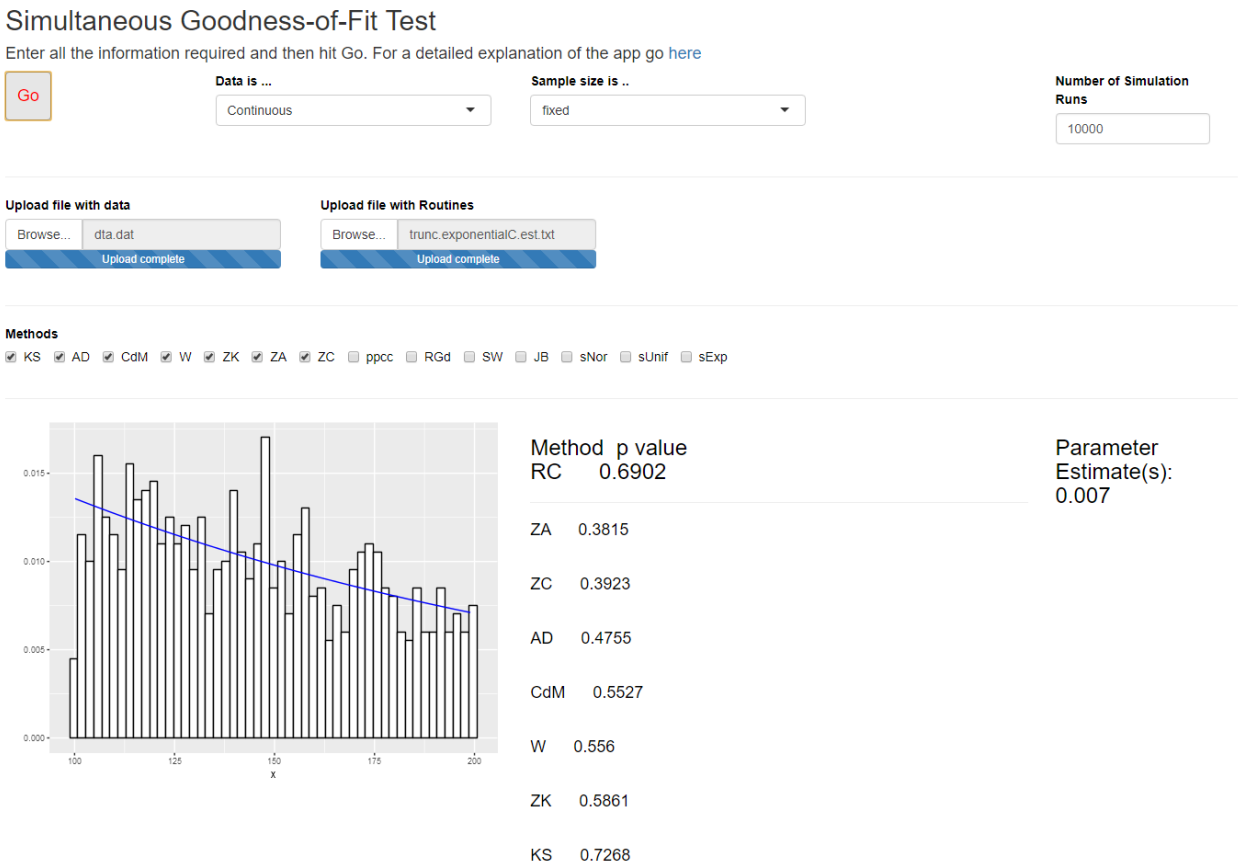
```
pnull <- function (x, param=1)
  A = 100; B = 200
  (exp(-A * param) - exp(-x * param))/(exp(-A * param) - exp(-B * param))

rnull <- function (n, param)
    A = 100; B = 200)
    x <- NULL
    repeat {
        x <- c(x, rexp(n, param))
        x <- x[x > A]
        x <- x[x < B]
        if (length(x) > n)
            break
    }
    x[1:n]
}

estimate <- function (x) {
    A = 100; B = 200
    n <- length(x)
    s <- sum(x)
    p <- n/s
    repeat {
        o <- p
        tmp1 <- exp(-o * A) - exp(-o * B)
        tmp2 <- B * exp(-o * B) - A * exp(-o * A)
        tmp3 <- A^2 * exp(-o * A) - B^2 * exp(-o * B)
        l1 <- n/o - s - n * tmp2/tmp1
        l2 <- (-n/o^2 - n * (tmp3 * tmp1 - tmp2^2)/tmp1^2)
        p <- o - l1/l2
        if (abs(p - o) < 0.001)
            break
    }
    p
}
```

C++ code:

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
NumericVector pnull(NumericVector x, double param) {
  int n=x.length();
  double A=100.0;
  double B=200.0;
  NumericVector y(n);
  y=(exp(-A*param)-exp(-x*param))/(exp(-A*param)-exp(-B*param));
  return y;
}
```

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
NumericVector rnull(int n, double param) {
  NumericVector y(n);
  double A=100.0;
  double B=200.0;
  for(int i=0; i<n; ++i) {
    do {
      y(i) = -log(runif(1)[0])/param;
    } while (y(i)<A | y(i)>B);
  }
  return y;
}
```

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
double estimate(NumericVector x) {
  int n=x.length();
  double A=100.0;
  double B=200.0;
  double s=sum(x);
  double p=double(n)/s;
  double o,tmp1,tmp2,tmp3,l1,l2;
  do {
    o=p;
    tmp1=exp(-o*A)-exp(-o*B);
    tmp2=B*exp(-o*B)-A*exp(-o*A);
    tmp3=A*A*exp(-o*A) - B*B*exp(-o*B);
    l1=n/o - s - n * tmp2/tmp1;
    l2=(-double(n)/o/o - n*(tmp3*tmp1-tmp2*tmp2)/tmp1/tmp1);
    p=o-l1/l2;
  } while ((p-o)*(p-o)>0.0000001);
  return p;
}
```



Example: Uniform [0,1], binned, random sample size

- Data

```
x <- runif(1000)
bins <- 0:50/50
counts <- hist(x, bins,plot=FALSE)$counts
write(c(bins[1], counts[1]), "shiny/dta.dat")
```

```
## Error in file(file, ifelse(append, "a", "w")): cannot open the connection
```

```
for(i in 2:50)
  write(c(bins[i], counts[i]), "shiny/dta.dat", append=TRUE)
```

```
## Error in file(file, ifelse(append, "a", "w")): cannot open the connection
```

```
write(bins[51], "shiny/dta.dat", append=TRUE)
```

```
## Error in file(file, ifelse(append, "a", "w")): cannot open the connection
```

Rcode:

```
pnull <- function(x) {
    A=0;B=1
    punif(x, A, B)
}
rnull <- function(n)  {
   A=0;B=1
   nbins=50
```

```
   bins <- A+(B-A)*0:nbins/nbins
   list(bins=bins, counts=hist(runif(n, A, B), bins, plot=FALSE)$counts)
}
```

C++ code:

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
NumericVector pnull(NumericVector x) {
  int n = x.length();
  NumericVector y(n);
  double A=0.0;
  double B=1.0;
  y=punif(x, A, B);
  return y;
}

#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
List rnull(int n) {
  double A=0.0;
  double B=1.0;
  double tmp;
  int i,j;
  int nbins=50;
  NumericVector bins(nbins+1);
  NumericVector x(nbins);
  for(i=0;i<nbins+1;++i) bins[i]=A+(B-A)*double(i)/nbins;
  for(i=0;i<nbins;++i) x[i]=0;
  for(j=0;j<n;++j) {
    tmp=A+(B-A)*runif(1)[0];
    i=0;
    do {
      ++i;
    } while (tmp>bins[i]);
    x[i-1]=x[i-1]+1;
  }
  return List::create(
    _["bins"]  = bins,
    _["counts"]  = x
  );
}
```

Simultaneous Goodness-of-Fit Test

Enter all the information required and then hit Go. For a detailed explanation of the app go here

Go

Data is ... Binned

Sample size is .. Poisson

Rate: 1000

Number of Simulation Runs 10000

Upload file with data

Browse... dta.dat

Upload complete

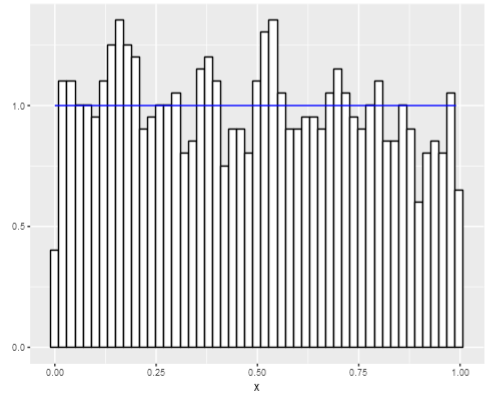
Upload file with Routines

Browse... uniform.binned.txt

Upload complete

Methods

KS AD CdM W ZK ZA ZC ppcc RGd SW JB sNor sUnif sExp



Method	p value
RC	0.2319
CdM	0.0962
AD	0.1084
KS	0.268
ZC	0.2841
ZA	0.289
W	0.3518
ZK	0.3882