This package provides facilities for performing and reporting different types of univariate goodness-of-fit statistical tests.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>2</td>
</tr>
<tr>
<td>FDist</td>
<td>4</td>
</tr>
<tr>
<td>FBar</td>
<td>5</td>
</tr>
<tr>
<td>KernelDensity</td>
<td>6</td>
</tr>
<tr>
<td>GofStat</td>
<td>7</td>
</tr>
<tr>
<td>GofFormat</td>
<td>14</td>
</tr>
</tbody>
</table>
Overview

This package contains tools for performing univariate goodness-of-fit (GOF) statistical tests. Methods for computing (or approximating) the distribution function $F(x)$ of certain GOF test statistics, as well as their complementary distribution function $\bar{F}(x) = 1 - F(x)$, are implemented in classes of package `probdist`. Tools for computing the GOF test statistics and the corresponding $p$-values, and for formating the results, are provided in classes `GofStat` and `GofFormat`.

We are concerned here with GOF test statistics for testing the hypothesis $H_0$ that a sample of $N$ observations $X_1, \ldots, X_N$ comes from a given univariate probability distribution $F$. We consider tests such as those of Kolmogorov-Smirnov, Anderson-Darling, Crâmer-von Mises, etc. These test statistics generally measure, in different ways, the distance between a continuous distribution function $F$ and the empirical distribution function (EDF) $\hat{F}_N$ of $X_1, \ldots, X_N$. They are also called EDF test statistics. The observations $X_i$ are usually transformed into $U_i = F(X_i)$, which satisfy $0 \leq U_i \leq 1$ and which follow the $U(0,1)$ distribution under $H_0$. (This is called the probability integral transformation.) Methods for applying this transformation, as well as other types of transformations, to the observations $X_i$ or $U_i$ are provided in `GofStat`.

Then the GOF tests are applied to the $U_i$ sorted by increasing order. The corresponding $p$-values are easily computed by calling the appropriate methods in the classes of package `probdist`. If a GOF test statistic $Y$ has a continuous distribution under $H_0$ and takes the value $y$, its (right) $p$-value is defined as $p = P[Y \geq y | H_0]$. The test usually rejects $H_0$ if $p$ is deemed too close to 0 (for a one-sided test) or too close to 0 or 1 (for a two-sided test).

In the case where $Y$ has a discrete distribution under $H_0$, we distinguish the right $p$-value $p_R = P[Y \geq y | H_0]$ and the left $p$-value $p_L = P[Y \leq y | H_0]$. We then define the $p$-value for a two-sided test as

$$ p = \begin{cases} 
  p_R, & \text{if } p_R < p_L \\
  1 - p_L, & \text{if } p_R \geq p_L \text{ and } p_L < 0.5 \\
  0.5 & \text{otherwise.}
\end{cases} \quad (1) $$

Why such a definition? Consider for example a Poisson random variable $Y$ with mean 1 under $H_0$. If $Y$ takes the value 0, the right $p$-value is $p_R = P[Y \geq 0 | H_0] = 1$. In the uniform case, this would obviously lead to rejecting $H_0$ on the basis that the $p$-value is too close to 1. However, $P[Y = 0 | H_0] = 1/e \approx 0.368$, so it does not really make sense to reject $H_0$ in this case. In fact, the left $p$-value here is $p_L = 0.368$, and the $p$-value computed with the above definition is $p = 1 - p_L \approx 0.632$. Note that if $p_L$ is very small, in this definition, $p$ becomes close to 1. If the left $p$-value was defined as $p_L = 1 - p_R = P[Y < y | H_0]$, this would also lead to problems. In the example, one would have $p_L = 0$ in that case.

A very common type of test in the discrete case is the chi-square test, which applies when the possible outcomes are partitioned into a finite number of categories. Suppose there are $k$ categories and that each observation belongs to category $i$ with probability $p_i$, for $0 \leq i < k$. 
If there are \( n \) independent observations, the expected number of observations in category \( i \) is \( e_i = np_i \), and the chi-square test statistic is defined as

\[
X^2 = \sum_{i=0}^{k-1} \frac{(o_i - e_i)^2}{e_i}
\]

(2)

where \( o_i \) is the actual number of observations in category \( i \). Assuming that all \( e_i \)'s are large enough (a popular rule of thumb asks for \( e_i \geq 5 \) for each \( i \)), \( X^2 \) follows approximately the chi-square distribution with \( k - 1 \) degrees of freedom \([12]\). The class \texttt{GofStat.OutcomeCategoriesChi2}, a nested class defined inside the \texttt{GofStat} class, provides tools to automatically regroup categories in the cases where some \( e_i \)'s are too small.

The class \texttt{GofFormat} contains methods used to format results of GOF test statistics, or to apply several such tests simultaneously to a given data set and format the results to produce a report that also contains the \( p \)-values of all these tests. A C version of this class is actually used extensively in the package TestU01, which applies statistical tests to random number generators \([9]\). The class also provides tools to plot an empirical or theoretical distribution function, by creating a data file that contains a graphic plot in a format compatible with a given software.
FDist

This class provides methods to compute (or approximate) the distribution functions of special types of goodness-of-fit test statistics.

```java
package umontreal.iro.lecuyer.gof;

public class FDist

    public static double kolmogorovSmirnovPlusJumpOne (int N, double a, double x)
    
    Similar to KolmogorovSmirnovPlusDist but for the case where the distribution function $F$ has a jump of size $a$ at a given point $x_0$, is zero at the left of $x_0$, and is continuous at the right of $x_0$. The Kolmogorov-Smirnov statistic is defined in that case as

$$D^+_N(a) = \sup_{a \leq u \leq 1} \left( \hat{F}_N(F^{-1}(u)) - u \right) = \max_{1+aN \leq j \leq N} \left( j/N - F(V(j)) \right). \quad (3)$$

where $V(1), \ldots, V(N)$ are the observations sorted by increasing order. The method returns an approximation of $P[D^+_N(a) \leq x]$ computed via

$$P[D^+_N(a) \leq x] = 1 - x \sum_{i=0}^{\lfloor N(1-a-x) \rfloor} \binom{N}{i} \left( \frac{i}{N} + x \right)^{i-1} \left( 1 - \frac{i}{N} - x \right)^{N-i}. \quad (4)$$

$$= x \sum_{j=0}^{\lfloor N(a+x) \rfloor} \binom{N}{j} \left( \frac{j}{N} - x \right)^j \left( 1 - \frac{j}{N} + x \right)^{N-j-1}. \quad (5)$$

The current implementation uses formula (5) when $N(x+a) < 6.5$ and $x+a < 0.5$, and uses (4) when $Nx \geq 6.5$ or $x+a \geq 0.5$. Restriction: $0 < a < 1.$

    public static double scan (int N, double d, double m)
    
    Returns $F(m)$, the distribution function of the scan statistic with parameters $N$ and $d$, evaluated at $m$. For a description of this statistic and its distribution, see scan, which computes its complementary distribution $\bar{F}(m) = 1 - F(m-1).$
This class is similar to FDist, except that it provides static methods to compute or approximate the complementary distribution function of $X$, which we define as $\bar{F}(x) = P[X \geq x]$, instead of $F(x) = P[X \leq x]$. Note that with our definition of $\bar{F}$, one has $\bar{F}(x) = 1 - F(x)$ for continuous distributions and $\bar{F}(x) = 1 - F(x-1)$ for discrete distributions over the integers.

```java
package umontreal.iro.lecuyer.gof;

public class FBar {
    public static double scan (int n, double d, int m)
    {
        // Return \( P[S_N(d) \geq m] \), where \( S_N(d) \) is the scan statistic defined as
        // \( S_N(d) = \sup_{0 \leq y \leq 1-d} \eta[y, y+d] \), where \( d \) is a constant in (0,1), \( \eta[y, y+d] \) is the number of observations falling inside the interval \([y, y+d]\), from a sample of \( N \) i.i.d. \( U(0,1) \) random variables. One has (see [1]),
        // \[ P[S_N(d) \geq m] \approx \left( \frac{m}{d} - N - 1 \right) b(m) + 2 \sum_{i=m}^{N} b(i) \]
        // \[ \approx 2(1 - \Phi(\theta \kappa)) + \theta \kappa \exp\left(-\frac{\theta^2 \kappa^2}{2}\right) \]
        // where \( \Phi \) is the standard normal distribution function.
        // \( b(i) = \binom{N}{i} d^i (1-d)^{N-i} \),
        // \( \theta = \sqrt{\frac{d}{1-d}} \),
        // \( \kappa = \frac{m}{d\sqrt{N}} - \sqrt{N} \).
        // For \( d \leq 1/2 \), (7) is exact for \( m > N/2 \), but only an approximation otherwise. The approximation (8) is good when \( Nd^2 \) is large or when \( d > 0.3 \) and \( N > 50 \). In other cases, this implementation sometimes use the approximation proposed by Glaz [5]. For more information, see [1] [5] [16]. The approximation returned by this function is generally good when it is close to 0, but is not very reliable when it exceeds, say, 0.4.
        // If \( m \leq (N+1)d \), the method returns 1. Else, if \( Nd \leq 10 \), it returns the approximation given by Glaz [5]. If \( Nd > 10 \), it computes (8) or (7) and returns the result if it does not exceed 0.4, otherwise it computes the approximation from [5], returns it if it is less than 1.0, and returns 1.0 otherwise. The relative error can reach 10% when \( Nd \leq 10 \) or when the returned value is less than 0.4. For \( m > Nd \) and \( Nd > 10 \), a returned value that exceeds 0.4 should be regarded as unreliable. For \( m = 3 \), the returned values are totally unreliable. (There may be an error in the original formulae in [5]).
        // Restrictions: \( N \geq 2 \) and \( d \leq 1/2 \).
    }
}
```
KernelDensity

This class provides methods to compute a kernel density estimator from a set of \( n \) individual observations \( x_0, \ldots, x_{n-1} \), and returns its value at \( m \) selected points. For details on how the kernel density is defined, and how to select the kernel and the bandwidth \( h \), see the documentation of class \KernelDensityGen in package \randvar.

```java
package umontreal.iro.lecuyer.gof;
import umontreal.iro.lecuyer.probdist.*;

public class KernelDensity

Methods

public static double[] computeDensity (EmpiricalDist dist,
    ContinuousDistribution kern,
    double h, double[] Y)

Given the empirical distribution \( \text{dist} \), this method computes the kernel density estimate at each of the \( m \) points \( Y[j], j = 0, 1, \ldots, (m - 1) \), where \( m \) is the length of \( Y \), the kernel is \( \text{kern.density(x)} \), and the bandwidth is \( h \). Returns the estimates as an array of \( m \) values.

public static double[] computeDensity (EmpiricalDist dist,
    ContinuousDistribution kern,
    double[] Y)

Similar to method \computeDensity above, but the bandwidth \( h \) is obtained from the method \KernelDensityGen.getBaseBandwidth(dist) in package \randvar.
```
This class provides methods to compute several types of EDF goodness-of-fit test statistics and to apply certain transformations to a set of observations. This includes the probability integral transformation $U_i = F(X_i)$, as well as the power ratio and iterated spacings transformations. Here, $U(0), \ldots, U(n-1)$ stand for $n$ observations $U_0, \ldots, U_{n-1}$ sorted by increasing order, where $0 \leq U_i \leq 1$.

Note: This class uses the Colt library.

```
package umontreal.iro.lecuyer.gof;
  import cern.colt.list.*;

public class GofStat

Transforming the observations

public static DoubleArrayList unifTransform (DoubleArrayList data,
  ContinuousDistribution dist)

Applies the probability integral transformation $U_i = F(V_i)$ for $i = 0, 1, \ldots, n-1$, where $F$ is a continuous distribution function, and returns the result as an array of length $n$. $V$ represents the $n$ observations contained in data, and $U$, the returned transformed observations. If data contains random variables from the distribution function dist, then the result will contain uniform random variables over $[0, 1]$.

public static DoubleArrayList unifTransform (DoubleArrayList data,
  DiscreteDistribution dist)

Applies the transformation $U_i = F(V_i)$ for $i = 0, 1, \ldots, n-1$, where $F$ is a discrete distribution function, and returns the result as an array of length $n$. $V$ represents the $n$ observations contained in data, and $U$, the returned transformed observations.

Note: If $V$ are the values of random variables with distribution function dist, then the result will contain the values of discrete random variables distributed over the set of values taken by dist, not uniform random variables over $[0, 1]$.

public static void diff (IntArrayList sortedData, IntArrayList spacings,
  int n1, int n2, int a, int b)

Assumes that the real-valued observations $U_0, \ldots, U_{n-1}$ contained in sortedData are already sorted in increasing order and computes the differences between the successive observations. Let $D$ be the differences returned in spacings. The difference $U_i - U_{i-1}$ is put in $D_i$ for $n1 < i \leq n2$, whereas $U_n1 - a$ is put into $D_{n1}$ and $b - U_n2$ is put into $D_{n2+1}$. The number of observations must be greater or equal than $n2$, we must have $n1 < n2$, and $n1$ and $n2$ are greater than 0. The size of spacings will be at least $n + 1$ after the call returns.

public static void diff (DoubleArrayList sortedData,
  DoubleArrayList spacings,
  int n1, int n2, double a, double b)

Same as method diff(IntArrayList,IntArrayList,int,int,int,int), but for the continuous case.
public static void iterateSpacings (DoubleArrayList data,
   DoubleArrayList spacings)
Applies one iteration of the *iterated spacings* transformation \[7, 15\]. Let \( U \) be the \( n \) observations contained into data, and let \( S \) be the spacings contained into spacings. Assumes that \( S[0..n] \) contains the *spacings* between \( n \) real numbers \( U_0, \ldots, U_{n-1} \) in the interval \([0, 1]\). These spacings are defined by
\[
S_i = U_{(i)} - U_{(i-1)}, \quad 1 < i < n,
\]
where \( U_{(0)} = 0 \), \( U_{(n-1)} = 1 \), and \( U_0, \ldots, U_{(n-1)} \), are the \( U_i \) sorted in increasing order. These spacings may have been obtained by calling *diff*. This method transforms the spacings into new spacings, by a variant of the method described in section 11 of \[11\] and also by Stephens \[15\]: it sorts \( S_0, \ldots, S_n \) to obtain \( S_0 \leq S_1 \leq S_2 \leq \cdots \leq S_n \), computes the weighted differences
\[
S_0 = (n + 1)S(0), \\
S_1 = n(S(1) - S(0)), \\
S_2 = (n - 1)(S(2) - S(1)), \\
\vdots \\
S_n = S(n) - S(n-1),
\]
and computes \( V_i = S_0 + S_1 + \cdots + S_i \) for \( 0 < i < n \). It then returns \( S_0, \ldots, S_n \) in \( S[0\ldots n] \) and \( V_1, \ldots, V_n \) in \( V[1\ldots n] \).

Under the assumption that the \( U_i \) are i.i.d. \( U(0, 1) \), the new \( S_i \) can be considered as a new set of spacings having the same distribution as the original spacings, and the \( V_i \) are a new sample of i.i.d. \( U(0, 1) \) random variables, sorted by increasing order.

This transformation is useful to detect clustering in a data set: A pair of observations that are close to each other is transformed into an observation close to zero. A data set with unusually clustered observations is thus transformed to a data set with an accumulation of observations near zero, which is easily detected by the Anderson-Darling GOF test.

public static void powerRatios (DoubleArrayList sortedData)
Applies the *power ratios* transformation \( W \) described in section 8.4 of Stephens \[15\]. Let \( U \) be the \( n \) observations contained into sortedData. Assumes that \( U \) contains \( n \) real numbers \( U_0, \ldots, U_{(n-1)} \) from the interval \([0, 1]\), already sorted in increasing order, and computes the transformations:
\[
U_i' = (U_{(i)} / U_{(i+1)})^{i+1}, \quad i = 0, \ldots, n-1,
\]
with \( U_{(n)} = 1 \). These \( U_i' \) are sorted in increasing order and put back in \( U[1\ldots n] \). If the \( U_{(i)} \) are i.i.d. \( U(0, 1) \) sorted by increasing order, then the \( U_i' \) are also i.i.d. \( U(0, 1) \).

This transformation is useful to detect clustering, as explained in *iterateSpacings*, except that here a pair of observations close to each other is transformed into an observation close to 1. An accumulation of observations near 1 is also easily detected by the Anderson-Darling GOF test.
Partitions for the chi-square tests

public static class OutcomeCategoriesChi2

This class helps managing the partitions of possible outcomes into categories for applying chi-square tests. It permits one to automatically regroup categories to make sure that the expected number of observations in each category is large enough. To use this facility, one must first construct an OutcomeCategoriesChi2 object by passing to the constructor the expected number of observations for each original category. Then, calling the method regroupCategories will regroup categories in a way that the expected number of observations in each category reaches a given threshold minExp. Experts in statistics recommend that minExp be always larger than or equal to 5 for the chi-square test to be valid. Thus, minExp = 10 is a safe value to use. After the call, nbExp gives the expected numbers in the new categories and loc[i] gives the relocation of category i, for each i. That is, loc[i] = j means that category i has been merged with category j because its original expected number was too small, and nbExp[i] has been added to nbExp[j] and then set to zero. In this case, all observations that previously belonged to category i are redirected to category j. The variable nbCategories gives the final number of categories, smin contains the new index of the lowest category, and smax the new index of the highest category.

public int nbCategories;
    Total number of categories.

public int smin;
    Minimum index for valid expected numbers in the array nbExp.

public int smax;
    Maximum index for valid expected numbers in the array nbExp.

public double[] nbExp;
    Expected number of observations for each category.

public int[] loc;
    loc[i] gives the relocation of the category i in the nbExp array.

public OutcomeCategoriesChi2 (double[] nbExp)

    Constructs an OutcomeCategoriesChi2 object using the array nbExp for the number of expected observations in each category. The smin and smax fields are set to 0 and (n - 1) respectively, where n is the length of array nbExp. The loc field is set such that loc[i] = i for each i. The field nbCategories is set to n.

public OutcomeCategoriesChi2 (double[] nbExp, int smin, int smax)

    Constructs an OutcomeCategoriesChi2 object using the given nbExp expected observations array. Only the expected numbers from the smin to smax (inclusive) indices will be considered valid. The loc field is set such that loc[i] = i for each i in the interval [smin, smax]. All loc[i] for i ≤ smin are set to smin, and all loc[i] for i ≥ smax are set to smax. The field nbCategories is set to (smax - smin + 1).
public OutcomeCategoriesChi2 (double[] nbExp, int[] loc,
   int smin, int smax, int nbCat)

Constructs an OutcomeCategoriesChi2 object. The field nbCategories is set to nbCat.

public void regroupCategories (double minExp)

Regroup categories as explained earlier, so that the expected number of observations in each category is at least minExp. We usually choose minExp = 10.

public String toString()

Provides a report on the categories.

Computing EDF test statistics

public static double chi2 (double[] nbExp, int[] count,
   int smin, int smax)

Computes and returns the chi-square statistic for the observations \(o_i\) in count[smin...smax], for which the corresponding expected values \(e_i\) are in nbExp[smin...smax]. Assuming that \(i\) goes from 1 to \(k\), where \(k = smax-smin+1\) is the number of categories, the chi-square statistic is defined as

\[
X^2 = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i}.
\] (9)

Under the hypothesis that the \(e_i\) are the correct expectations and if these \(e_i\) are large enough, \(X^2\) follows approximately the chi-square distribution with \(k - 1\) degrees of freedom. If some of the \(e_i\) are too small, one can use OutcomeCategoriesChi2 to regroup categories.

public static double chi2 (OutcomeCategoriesChi2 cat, int[] count)

Computes and returns the chi-square statistic for the observations \(o_i\) in count, for which the corresponding expected values \(e_i\) are in cat. This assumes that cat.regroupCategories has been called before to regroup categories in order to make sure that the expected numbers in each category are large enough for the chi-square test.

public static double chi2 (IntArrayList data, DiscreteDistributionInt dist,
   int smin, int smax, double minExp, int[] numCat)

Computes and returns the chi-square statistic for the observations stored in data, assuming that these observations follow the discrete distribution dist. For dist, we assume that there is one set \(S = \{a, a+1, \ldots, b-1, b\}\), where \(a < b\) and \(a \geq 0\), for which \(p(s) > 0\) if \(s \in S\) and \(p(s) = 0\) otherwise.

Generally, it is not possible to divide the integers in intervals satisfying \(nP(a_0 \leq s < a_1) = nP(a_1 \leq s < a_2) = \cdots = nP(a_{j-1} \leq s < a_j)\) for a discrete distribution, where \(n\) is the sample size, i.e., the number of observations stored into data. To perform a general chi-square test, the method starts from smin and finds the first non-negligible probability \(p(s) \geq \epsilon\), where \(\epsilon = DiscreteDistributionInt.EPSILON\). It uses smax to allocate an array storing the number of expected observations (np(s)) for each \(s \geq smin\). Starting from \(s = smin\), the np(s)
terms are computed and the allocated array grows if required until a negligible probability term is found. This gives the number of expected elements for each category, where an outcome category corresponds here to an interval in which sample observations could lie. The categories are regrouped to have at least \( \text{minExp} \) observations per category. The method then counts the number of samples in each categories and calls \( \text{chi2} \) to get the chi-square test statistic. If \( \text{numCat} \) is not null, the number of categories after regrouping is returned in \( \text{numCat}[0] \). The number of degrees of freedom is equal to \( \text{numCat}[0]-1 \). We usually choose \( \text{minExp} = 10 \).

```java
public static double chi2Equal (double nbExp, int[] count,
    int smin, int smax)
```

Similar to \( \text{chi2} \), except that the expected number of observations per category is assumed to be the same for all categories, and equal to \( \text{nbExp} \).

```java
public static double chi2Equal (DoubleArrayList data, double minExp)
```

Computes the chi-square statistic for a continuous distribution. Here, the equiprobable case can be used. Assuming that \( \text{data} \) contains observations coming from the uniform distribution, the \([0, 1]\) interval is divided into \( 1/p \) subintervals, where \( p = \text{minExp}/n \), \( n \) being the sample size, i.e., the number of observations stored in \( \text{data} \). For each subinterval, the method counts the number of contained observations and the chi-square statistic is computed using \( \text{chi2Equal} \). We usually choose \( \text{minExp} = 10 \).

```java
public static double chi2Equal (DoubleArrayList data)
```

Equivalent to \( \text{chi2Equal (data, 10)} \).

```java
public static int scan (DoubleArrayList sortedData, double d)
```

Computes and returns the scan statistic \( S_n(d) \), defined in (6). Let \( U \) be the \( n \) observations contained into \( \text{sortedData} \). The \( n \) observations in \( U[0..n-1] \) must be real numbers in the interval \([0, 1]\), sorted in increasing order. (See \( \text{FBar.scan} \) for the distribution function of \( S_n(d) \)).

```java
public static double cramerVonMises (DoubleArrayList sortedData)
```

Computes and returns the Cramér-von Mises statistic \( W^2_n \) (see [4, 13, 14]), defined by

\[
W^2_n = \frac{1}{12n} + \sum_{j=0}^{n-1} \left( U_j - \frac{(j + 0.5)}{n} \right)^2,
\]

assuming that \( \text{sortedData} \) contains \( U(0), \ldots, U(n-1) \) sorted in increasing order.

```java
public static double watsonG (DoubleArrayList sortedData)
```

Computes and returns the Watson statistic \( G_n \) (see [17, 3]), defined by

\[
G_n = \sqrt{n} \max_{0 \leq j \leq n-1} \left\{ (j + 1)/n - U_j + \frac{U_n}{2} - 1/2 \right\}
\]

\[
= \sqrt{n} \left( D_n^+ + \frac{U_n}{2} - 1/2 \right),
\]

where \( U_n \) is the average of the observations \( U(j) \), assuming that \( \text{sortedData} \) contains the sorted \( U(0), \ldots, U(n-1) \).
public static double watsonU (DoubleArrayList sortedData)
Computes and returns the Watson statistic $U_n^2$ (see [4] [13] [14]), defined by

$$W_n^2 = \frac{1}{12n} + \sum_{j=0}^{n-1} \left\{ U_j - \frac{(j + 0.5)}{n} \right\}^2,$$

(12)

$$U_n^2 = W_n^2 - n (\overline{U}_n - 1/2)^2. \quad (13)$$

where $\overline{U}_n$ is the average of the observations $U_j$, assuming that sortedData contains the sorted $U(0), \ldots, U(n-1)$.

public static double EPSILONAD = Num.DBL_EPSILON/2;
Used by andersonDarling. Num.DBL_EPSILON is usually $2^{-52}$.

public static double andersonDarling (DoubleArrayList sortedData)
Computes and returns the Anderson-Darling statistic $A_n^2$ (see method andersonDarling).

public static double andersonDarling (double[] sortedData)
Computes and returns the Anderson-Darling statistic $A_n^2$ (see [10] [14] [2]), defined by

$$A_n^2 = -n - \frac{1}{n} \sum_{j=0}^{n-1} \left\{ (2j + 1) \ln(U(j)) + (2n - 1 - 2j) \ln(1 - U(j)) \right\},$$

assuming that sortedData contains $U(0), \ldots, U(n-1)$ sorted in increasing order.

When computing $A_n^2$, all observations $U_i$ are projected on the interval $[\epsilon, 1 - \epsilon]$ for some $\epsilon > 0$, in order to avoid numerical overflow when taking the logarithm of $U_i$ or $1 - U_i$. The variable EPSILONAD gives the value of $\epsilon$.

public static double[] andersonDarling (double[] data, ContinuousDistribution dist)
Computes the Anderson-Darling statistic $A_n^2$ and the corresponding p-value $p$. The $n$ (unsorted) observations in data are assumed to be independent and to come from the continuous distribution dist. Returns the 2-elements array $[A_n^2, p]$.

public static double[] kolmogorovSmirnov (double[] sortedData)
Computes the Kolmogorov-Smirnov (KS) test statistics $D_n^+$, $D_n^-$, and $D_n$ (see method kolmogorovSmirnov). Returns the array $[D_n^+, D_n^-, D_n]$.

public static double[] kolmogorovSmirnov (DoubleArrayList sortedData)
Computes the Kolmogorov-Smirnov (KS) test statistics $D_n^+$, $D_n^-$, and $D_n$ defined by

$$D_n^+ = \max_{0 \leq j \leq n-1} \left( (j + 1)/n - U(j) \right), \quad (14)$$

$$D_n^- = \max_{0 \leq j \leq n-1} \left( U(j) - j/n \right), \quad (15)$$

$$D_n = \max(D_n^+, D_n^-). \quad (16)$$
and returns an array of length 3 that contains \([D^+_n, D^-_n, D_n]\). These statistics compare the empirical distribution of \(U(1), \ldots, U(n)\), which are assumed to be in `sortedData`, with the uniform distribution over \([0, 1]\).

```java
public static void kolmogorovSmirnov (double[] data,
                                    ContinuousDistribution dist,
                                    double[] sval,
                                    double[] pval)
```

Computes the Kolmogorov-Smirnov (KS) test statistics and their \(p\)-values. This is to compare the empirical distribution of the (unsorted) observations in `data` with the theoretical distribution `dist`. The KS statistics \(D^+_n, D^-_n\) and \(D_n\) are returned in `sval[0]`, `sval[1]`, and `sval[2]` respectively, and their corresponding \(p\)-values are returned in `pval[0]`, `pval[1]`, and `pval[2]`.

```java
public static double[] kolmogorovSmirnovJumpOne (DoubleArrayList sortedData,
                                                  double a)
```

Compute the KS statistics \(D^+_n(a)\) and \(D^-_n(a)\) defined in the description of the method `FDist.kolmogorovSmirnovPlusJumpOne`, assuming that \(F\) is the uniform distribution over \([0, 1]\) and that \(U(1), \ldots, U(n)\) are in `sortedData`. Returns the array \([D^+_n, D^-_n]\).

```java
public static double pDisc (double pL, double pR)
```

Computes a variant of the \(p\)-value \(p\) whenever a test statistic has a discrete probability distribution. This \(p\)-value is defined as follows:

\[
\begin{align*}
p_L &= P[Y \leq y] \\
p_R &= P[Y \geq y] \\
p &= \begin{cases} 
p_R, & \text{if } p_R < p_L \\ 1 - p_L, & \text{if } p_R \geq p_L \text{ and } p_L < 0.5 \\ 0.5 & \text{otherwise.}
\end{cases}
\end{align*}
\]

The function takes \(p_L\) and \(p_R\) as input and returns \(p\).
**GofFormat**

This class contains methods used to format results of GOF test statistics, or to apply a series of tests simultaneously and format the results. It is in fact a translation from C to Java of a set of functions that were specially written for the implementation of TestU01, a software package for testing uniform random number generators [9].

Strictly speaking, applying several tests simultaneously makes the $p$-values “invalid” in the sense that the probability of having at least one $p$-value less than 0.01, say, is larger than 0.01. One must therefore be careful with the interpretation of these $p$-values (one could use, e.g., the Bonferroni inequality [8]). Applying simultaneous tests is convenient in some situations, such as in screening experiments for detecting statistical deficiencies in random number generators. In that context, rejection of the null hypothesis typically occurs with extremely small $p$-values (e.g., less than $10^{-15}$), and the interpretation is quite obvious in this case.

The class also provides tools to plot an empirical or theoretical distribution function, by creating a data file that contains a graphic plot in a format compatible with the software specified by the environment variable `graphSoft`. NOTE: see also the more recent package `charts`.

Note: This class uses the Colt library.

```java
package umontreal.iro.lecuyer.gof;
import cern.colt.list.);

class GofFormat

public static final int GNUPLOT
    Data file format used for plotting functions with Gnuplot.

public static final int MATHEMATICA
    Data file format used for creating graphics with Mathematica.

public static int graphSoft = GNUPLOT;
    Environment variable that selects the type of software to be used for plotting the graphs of functions. The data files produced by `graphFunc` and `graphDistUnif` will be in a format suitable for this selected software. The default value is GNUPLOT. To display a graphic in file `f` using `gnuplot`, for example, one can use the command “plot f with steps, x with lines” in `gnuplot`.

public static String drawCdf (ContinuousDistribution dist, double a, double b, int m, String desc)
    Formats data to plot the graph of the distribution function $F$ over the interval $[a,b]$, and returns the result as a `String`. The method `dist.cdf(x)` returns the value of $F$ at $x$. The
```
String desc gives a short caption for the graphic plot. The method computes the $m + 1$ points $(x_i, F(x_i))$, where $x_i = a + i(b - a)/m$ for $i = 0, 1, \ldots, m$, and formats these points into a String in a format suitable for the software specified by graphSoft. NOTE: see also the more recent class ContinuousDistChart.

public static String drawDensity (ContinuousDistribution dist, double a, double b, int m, String desc)

Formats data to plot the graph of the density $f(x)$ over the interval $[a,b]$, and returns the result as a String. The method dist.density(x) returns the value of $f(x)$ at $x$. The String desc gives a short caption for the graphic plot. The method computes the $m + 1$ points $(x_i, f(x_i))$, where $x_i = a + i(b - a)/m$ for $i = 0, 1, \ldots, m$, and formats these points into a String in a format suitable for the software specified by graphSoft. NOTE: see also the more recent class ContinuousDistChart.

public static String graphDistUnif (DoubleArrayList data, String desc)

Formats data to plot the empirical distribution of $U(1), \ldots, U(N)$, which are assumed to be in data[0...N-1], and to compare it with the uniform distribution. The $U(i)$ must be sorted. The two endpoints (0,0) and (1,1) are always included in the plot. The string desc gives a short caption for the graphic plot. The data is printed in a format suitable for the software specified by graphSoft. NOTE: see also the more recent class EmpiricalChart.

Computing and printing $p$-values for EDF test statistics

public static double EPSILONP = 1.0E-15;

Environment variable used in formatp0 to determine which $p$-values are too close to 0 or 1 to be printed explicitly. If EPSILONP = $\epsilon$, then any $p$-value less than $\epsilon$ or larger than $1 - \epsilon$ is not written explicitly; the program simply writes “eps” or “1-eps”. The default value is $10^{-15}$.

public static double SUSPECTP = 0.01;

Environment variable used in formatp1 to determine which $p$-values should be marked as suspect when printing test results. If SUSPECTP = $\alpha$, then any $p$-value less than $\alpha$ or larger than $1 - \alpha$ is considered suspect and is “singled out” by formatp1. The default value is 0.01.

public static String formatp0 (double p)

Returns the $p$-value $p$ of a test, in the format “1 - $p$” if $p$ is close to 1, and $p$ otherwise. Uses the environment variable EPSILONP and replaces $p$ by $\epsilon$ when it is too small.

public static String formatp1 (double p)

Returns the string “p-value of test : ”, then calls formatp0 to print $p$, and adds the marker “****” if $p$ is considered suspect (uses the environment variable SUSPECTP for this).

public static String formatp2 (double x, double p)

Returns $x$ on a single line, then go to the next line and calls formatp1.
public static String formatp3 (String testName, double x, double p)
Formats the test statistic x for a test named testName with p-value p. The first line of the returned string contains the name of the test and the statistic whereas the second line contains its p-value. The formated values of x and p are aligned.

public static String formatChi2 (int k, int d, double chi2)
Computes the p-value of the chi-square statistic chi2 for a test with k intervals. Uses d decimal digits of precision in the calculations. The result of the test is returned as a string. The p-value is computed using pDisc.

public static String formatKS (int n, double dp, double dm, double d)
Computes the p-values of the three Kolmogorov-Smirnov statistics $D^+_N$, $D^-_N$, and $D_N$, whose values are in dp, dm, d, respectively, assuming a sample of size n. Then formats these statistics and their p-values using formatp2 for each one.

public static String formatKS (DoubleArrayList data, ContinuousDistribution dist)
Computes the KS test statistics to compare the empirical distribution of the observations in data with the theoretical distribution dist and formats the results. See also method GofStat.kolmogorovSmirnov(double[],ContinuousDistribution,double[],double[]).

public static String formatKSJumpOne (int n, double a, double dp)
Similar to formatKS, but for the KS statistic $D^+_N(a)$ defined in (3). Writes a header, computes the p-value and calls formatp2.

public static String formatKSJumpOne (DoubleArrayList data, ContinuousDistribution dist, double a)
Similar to formatKS, but for $D^+_N(a)$ defined in (3).

Applying several tests at once and printing results

Higher-level tools for applying several EDF goodness-of-fit tests simultaneously are offered here. The environment variable activeTests specifies which tests in this list are to be performed when asking for several simultaneous tests via the functions activeTests, formatActiveTests, etc.

    public static final int KSP = 0;
    Kolmogorov-Smirnov+ test

    public static final int KSM = 1;
    Kolmogorov-Smirnov− test

    public static final int KS = 2;
    Kolmogorov-Smirnov test
public static final int AD = 3;
   Anderson-Darling test

public static final int CM = 4;
   Cramér-von Mises test

public static final int WG = 5;
   Watson G test

public static final int WU = 6;
   Watson U test

public static final int MEAN = 7;
   Mean

public static final int COR = 8;
   Correlation

public static final int NTESTTYPES = 9;
   Total number of test types

public static final String[] TESTNAMES
   Name of each testType test. Could be used for printing the test results, for example.

public static boolean[] activeTests
   The set of EDF tests that are to be performed when calling the methods activeTests,
   formatActiveTests, etc. By default, this set contains KSP, KSM, and AD. Note: MEAN and
   COR are always excluded from this set of active tests.

public static void tests (DoubleArrayList sortedData, double[] sVal)
   Computes all EDF test statistics enumerated above (except COR) to compare the empirical
   distribution of \( U(0), \ldots, U(N-1) \) with the uniform distribution, assuming that these
   sorted observations are in sortedData. If \( N > 1 \), returns sVal with the values of the
   KS statistics \( D^+ \), \( D^- \) and \( D_N \), of the Cramér-von Mises statistic \( W^2_N \), Watson’s \( G_N \) and
   \( U^2_N \), Anderson-Darling’s \( A^2_N \), and the average of the \( U_i \)’s, respectively. If \( N = 1 \), only puts
   \( 1 - \text{sortedData.get(0)} \) in sVal[KSP]. Calling this method is more efficient than computing
   these statistics separately by calling the corresponding methods in GofStat.

public static void tests (DoubleArrayList data, ContinuousDistribution dist, double[] sVal)
   The observations \( V \) are in data, not necessarily sorted, and their empirical distribution is
   compared with the continuous distribution dist. If \( N = 1 \), only puts data.get(0) in
   sVal[MEAN], and \( 1 - \text{dist.cdf(data.get(0))} \) in sVal[KSP].

public static void activeTests (DoubleArrayList sortedData, double[] sVal, double[] pVal)
   Computes the EDF test statistics by calling tests, then computes the \( p \)-values of those that
   currently belong to activeTests, and return these quantities in sVal and pVal, respectively.
Assumes that $U_{(0)}, \ldots, U_{(N-1)}$ are in sortedData and that we want to compare their empirical distribution with the uniform distribution. If $N = 1$, only puts $1 - \text{sortedData.get}(0)$ in sVal[KSP], pVal[KSP], and pVal[MEAN].

```java
public static void activeTests (DoubleArrayList data, ContinuousDistribution dist, double[] sVal, double[] pVal)
```

The observations are in data, not necessarily sorted, and we want to compare their empirical distribution with the distribution dist. If $N = 1$, only puts data.get(0) in sVal[MEAN], and $1 - \text{dist.cdf(data.get(0))}$ in sVal[KSP], pVal[KSP], and pVal[MEAN].

```java
public static String formatActiveTests (int n, double[] sVal, double[] pVal)
```

Gets the $p$-values of the active EDF test statistics, which are in activeTests. It is assumed that the values of these statistics and their $p$-values are already computed, in sVal and pVal, and that the sample size is $n$. These statistics and $p$-values are formatted using formatp2 for each one. If $n=1$, prints only pVal[KSP] using formatp1.

```java
public static String iterSpacingsTests (DoubleArrayList sortedData, int k, boolean printval, boolean graph, PrintWriter f)
```

Repeats the following $k$ times: Applies the GofStat.iterateSpacings transformation to the $U_{(0)}, \ldots, U_{(N-1)}$, assuming that these observations are in sortedData, then computes the EDF test statistics and calls activeTests after each transformation. The function returns the original array sortedData (the transformations are applied on a copy of sortedData). If printval = true, stores all the values into the returned String after each iteration. If graph = true, calls graphDistUnif after each iteration to print to stream $f$ the data for plotting the distribution function of the $U_i$.

```java
public static String iterPowRatioTests (DoubleArrayList sortedData, int k, boolean printval, boolean graph, PrintWriter f)
```

Similar to iterSpacingsTests, but with the GofStat.powerRatios transformation.
References


