13 Statistical description of data

Functions:

moment: calculate the moments of a data set
mdian1: calcuate the median of a data set by sorting
mdian2: calcuate the median of a data set iteratively
ttest: Student's t-test for the difference of means
avevar: calcuate the mean and variance of a data set
tutest: Student's t-test with unequal variances
fptest: Student's t-test with paired data
ftest: F-test for difference of variances
chsone: chi-square test for difference between data and model
chstwo: chi-square test for difference between two data sets
ksone: Kolmorgorov-Smirnov test of data against model
kstwo: Kolmorgorov-Smirnov test of two data sets
probks: Kolmorgorov-Smirnov probability function
cntab1: contingency table analysis using chi-square
cntab2: contingency table analysis using entropy measure
pearsn: Pearson's correlation between two data sets
spear: Spearman's rank correlation between two data sets
crank: replace array elements by their ranks
kendl1: Kendal's tau correlation between two data sets
kendl2: contingency table analysis using Kendall's tau
smooft: smooth data using FFT

---

moment

Description:

This function calculates the moments of a data set.

moment(data) ⟶ [ave, adev, sdev, var, skew, curt]

Where:

data is a real list being the data set,
ave is a floating point number being the mean of the data set,
adev is a floating point number being the average deviation,
sdev is a floating point number being the standard deviation,
var is a floating point number being the variance,
skew is a floating point number being the skewness,
curt is a floating point number being the kurtosis.

Example:

The moments of two data sets are computed using `moment`.

```
-> moment([1,2,3,4])?
[2.5000000, 1.0000000, 1.2909944, 1.6666667, 0.0000000, -2.0775000]
-> data:[1,2,3,4,5,6,7,8,9,10,11,12,13,14,13,12,11,10,9,8,7,6,5,4,3,2,1];
-> moment(data)?
[7.2592593, 3.3799726, 3.9767845, 15.8148148, 0.0133791, -1.3249498]
```

---

mdian1

Description:

This function calculates the median of a data set by sorting.

mdian1(x) ⟶ xmed

Where:

x is a real list representing the data set,
xmed is a floating point number being the median of the data set.

Example:

The median of a data set is computed using `mdian1`.

```
-> data:[1,2,3,4,5,6,7,8,9,10,11,12,13,14,13,12,11,10,9,8,7,6,5,4,3,2,1];
-> mdian1(data)?
7.0000000
-> mdian1([2,3,4,5,5,8,34])?
5.0000000
```

---

mdian2

Description:

This function computes the median of a data set using an iterative method.

$$\mathbf{mdian2(x)} \longrightarrow \mathbf{xmed}$$

Where:

x is a real list being the data set,
xmed is the median of the data set.

Pre-variables:

mdian2_big [1.0d+30] a floating point number,
mdian2_afc [1.5d0] a floating point number used to optimise the size of the smoothing constant at each iteration,
mdian2_amp [1.5d0] a floating point number being an overconvergence factor.

Example:

The medians of two data sets are computed using `mdian2`. Compare with results for same data sets `mdian1` above.

```
-> data:[1,2,3,4,5,6,7,8,9,10,11,12,13,14,13,12,11,10,9,8,7,6,5,4,3,2,1];
-> mdian2(data)?
7.0000000
-> mdian2([2,3,4,5,5,8,34])?
4.0000000
```

ttest

Description:

This function applies the Student's t-test for the difference of two means.

ttest(data1, data2) ⟶ [t, prob]

Where:

data1 is a real list,
data2 is a real list,
t is a floating point number being the value of the Student's t-statistic for the data sets *data*1 and *data*2,
prob is a floating point number representing the significance of the statistic t.

Example:

The function `ttest` is applied to two data sets.

```
-> data1 : [1,2,3,4,5,4,3,2,1];
-> data2 : [3,4,5,6,7,6,5,4,3];
-> ttest(data1,data2)?
[-3.0425553, 0.0077586]
```

---

avevar

Description:

This function calculates the mean and variance of a data set.

avevar(data) ⟶ [ave, var]

Where:

data is a real list,
ave is a floating point number representing the mean of *data*,
var is a floating point number representing the variance of *data*.

Example:

The function `avevar` is applied to two data sets. Compare with the results for same data sets in `moment` above.

```
-> avevar([1,2,3,4])?
[2.5000000, 1.6666667]
-> data:[1,2,3,4,5,6,7,8,9,10,11,12,13,14,13,12,11,10,9,8,7,6,5,4,3,2,1];
-> avevar(data)?
[7.2592593, 15.8148148]
```

---

tutest

Description:

This function applies the Student's t-test to two data sets with possibly unequal variances.

$$\textbf{tutest} \longrightarrow [\textbf{t}, \textbf{prob}]$$

Where:

data1 is a real list,
data2 is a real list,
t is a floating point number being the value of the Student's t statistic for the data sets $data1$ and $data2$,
prob is a floating point number representing the significance of the statistic t.

Example:

The function `tutest` is used to compare two data sets. Compare with the results for same data sets in `ttest` above.

```
-> data1 : [1,2,3,4,5,4,3,2,1];
-> data2 : [3,4,5,6,7,6,5,4,3];
-> tutest(data1,data2)?
[-3.0425553, 0.0077586]
```

tptest

Description:

This function applies the Student's t-test to two data sets with paired data.

tptest(data1, data2) ⟶ [t, prob]

Where:

data1 is a real list,
data2 is a real list paired with $data1$,
t is a floating point number being the value of the Student's t statistic for the data sets $data1$ and $data2$,
prob is a floating point number representing the significance of the statistic t.

Example:

The function `tptest` is used to compare two data sets.

```
-> data1 : [1,2,3,4,5,4,3,2,1];
-> data2 : [3,4,5,6,7,6,5,4,3];
-> lld("sp13", "nr13");
-> tptest(data1,data2)?
Senpack error:  zero joint deviation sd in tptest
-> tptest(listn(1,100), random_list(0,40,100))?
[10.1616249, 0.0000000]
```

---

ftest

Description:

This funtion applies the F-test for the difference of variances.

ftest(data1, data2) ⟶ [f, prob]

Where:

data1 is a real list,
data2 is a real list,
f is a floating point number representing the f statistic for $data1$ and $data2$,
prob is a floating point number representing the significance of f.

Example:

The function `ftest` is used to compare two data sets with the same variance.

```
-> data1 : [1,2,3,4,5,4,3,2,1];
-> data2 : [3,4,5,6,7,6,5,4,3];
-> ftest(data1,data2)?
[1.0000000, 1.0000000]
```

---

chsone

Description:

This funtion applies the Chi-square test for the difference between a data set and a model.

chsone(bins, ebins, knstrn) ⟶ [df, chsq, prob]

Where:

bins is a real list containing an observed numbers of events,
ebins is a real list containing the expected numbers of events,
kstrn is an integer being the number of constraints,
df is an integer being the number of degrees of freedom,
chsq is a floating point number being the chi-square statistic,
prob is a floating point number representing the significance of *chsq*.

Example:

The function `chsone` is used to apply the Chi-squared test to two data sets.

```
-> bins : [1,2,3,6,5,4,7,8];
-> ebins : [1,2,3,4,5,6,7,8];
-> chsone(bins,ebins,1)?
[6.0000000, 1.6666667, 0.9476655]
-> chsone(bins,ebins,3)?
[4.0000000, 1.6666667, 0.7967634]
-> chsone(bins,ebins,5)?
[2.0000000, 1.6666667, 0.4345982]
```

chstwo

Description:

This function applies the chi-square test for the difference between two data sets.

chstwo(bins1, bins2, knstrn) ⟶ [df, chsq, prob]

Where:

bins1 is a real list containing binned data,
bins2 is a real list containing binned data,
knstrn is an integer representing the number of additional constraints,
df is an integer being the number of degrees of freedom,
chsq is a floating point number being the chi-square statistic,
prob is a floating point number representing the significance of *chsq*.

Example:

Compare with results for same data set in `chsone` above.

```
-> bins1 : [1,2,3,6,5,4,7,8];
-> bins2 : [1,2,3,4,5,6,7,8];
-> chstwo(bins1,bins2,1)?
[6.0000000, 0.8000000, 0.9920737]
-> chstwo(bins1,bins2,3)?
[4.0000000, 0.8000000, 0.9384481]
-> chstwo(bins1,bins2,5)?
[2.0000000, 0.8000000, 0.6703200]
```

ksone

Description:

This function applies the Kolmogorov-Smirnov test for the difference between a data set and a model.

ksone(data, func) ⟶ [d, prob]

Where:

data is a real list,
func is an increasing real valued function of a real variable with range $[0, 1]$,
d is a floating point number being the Kolmogorov-Smirnov statistic,
prob is a floating point number representing the significance of d.

Example:

Take the error function which is the cumulative distribution function for the normal distribution.

```
-> f(x) := 1-exp(-x*1.0);
-> df(x) := exp(-1.0*x);
-> data : [9,9];
-> Should get approx. 1.0 for the K-S statistic #
-> ksone(data,f)?
[0.9998766, 0.0366672]
-> data : map(df,0.01*listn(0,1000));
-> Significance should be approx. 0.0 #
-> ksone(data,f)?
[0.6753375, 0.0000000]
```

kstwo

Description:

This function applies the Kolmogorv-Smirnov test to two data sets.

$$\mathbf{kstwo(data1,\ data2)} \longrightarrow \mathbf{[d,\ prob]}$$

Where:

data1 is a real list,
data2 is a real list,
d is a floating point number being the Kolmogorov-Smirnov statistic,
prob is a floating point number representing the significance level for the null hypothesis that the data sets are drawn from the same distribution.

Post-variables:

kstwo_data1 is a floating point list,
kstwo_data2 is a floating point list.

Example:

The function `kstwo` is applied to two data sets arising from the cumumative normal distribution.

```
-> df(x) := exp(-1.0*x);
-> data1 : map(df,[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0])?
[0.9048374, 0.8187308, 0.7408182, 0.6703200, 0.6065307, 0.5488116,
0.4965853, 0.4493290, 0.4065697, 0.3678795]
-> data2 : map(df,[0.05,0.15,0.25,0.35,0.45,0.55,0.65,0.75,0.85,0.95])?
[0.9512295, 0.8607080, 0.7788008, 0.7046881, 0.6376281, 0.5769498,
0.5220458, 0.4723665, 0.4274149, 0.3867410]
-> kstwo(data1,data2)?
[0.1000000, 1.0000000]
```

probks

Description:

This is the Kolmogorov-Smirnov probability function Q_{KS}.

$$\textbf{probks(alam)} \longrightarrow \textbf{value}$$

Where:

alam is a real number,
value is a floating point number being the value of Q_{KS} at *alam*.

Pre-variables:

probks_eps1 [0.001d0] a floating point number,
probks_eps2 [1.0d-8] a floating point number.

Example:

We have no independant check for `probks`, so graph it instead!

```
-> unbind(x,lam,value);
-> vals : [lam,value];
-> for i from 0 by 0.1 to 2 do sprog(lam : i+0.1, value : probks(lam),
     bar : "", for j to trunc(40*value) do bar : string_join(bar,"*"),
     print(insert_after(vals,bar)));
[0.1000000, 1.0000000, "****************************************"]
[0.2000000, 1.0000000, "***************************************"]
[0.3000000, 0.9999907, "***************************************"]
[0.4000000, 0.9971923, "***************************************"]
[0.5000000, 0.9639452, "**************************************"]
[0.6000000, 0.8642827, "**********************************"]
[0.7000000, 0.7112351, "****************************"]
[0.8000001, 0.5441423, "*********************"]
[0.9000001, 0.3927306, "***************"]
[1.0000001, 0.2699995, "**********"]
[1.1000001, 0.1777181, "*******"]
[1.2000002, 0.1122496, "****"]
[1.3000002, 0.0680922, "**"]
[1.4000002, 0.0396818, "*"]
[1.5000002, 0.0222179, ""]
[1.6000003, 0.0119520, ""]
[1.7000003, 0.0061774, ""]
[1.8000003, 0.0030676, ""]
[1.9000003, 0.0014636, ""]
[2.0000002, 0.0006709, ""]
```

cntab1

Description:

This function performs a contingency table analysis using chi-square.

cntab1(nn) $\longrightarrow$ [chisq, df, prob, cramrv, ccc]

Where:

nn is an integer matrix representing a two dimensional contingency table,
chisq is a floating point number being the chi-square statistic,
df is an integer being the number of degrees of freedom,
prob is the significance level of the chi-square statistic,
cramrv is a floating point number representing measure of association Cramer's V,
ccc is a floating point number representing the contingency coefficient C measure of association.

Pre-variables:

cntab1_tiny [1.0d-30] is a floating point number.

Example:

Contingency Table for `cntab1` and `cntab2` is set up in the matrix `nn`. The columns represent months January through December. The rows represent the types of deaths in 1979 being motor vehicle, falls, drowning, fires, choking, fire-arms, poisons, gas-poisons and other.

```
-> nn : matrix(
  [3298, 3304, 4241, 4291, 4594, 4710, 4914, 4942, 4861, 4914, 4563, 4892],
  [1150, 1034, 1089, 1126, 1142, 1100, 1112, 1099, 1114, 1079,  999, 1181],
  [180,  190,  370,  530,  800, 1130, 1320,  990,  580,  320,  250,  212],
  [874,  768,  630,  516,  385,  324,  277,  272,  271,  381,  533,  760],
  [299,  264,  258,  247,  273,  269,  251,  269,  271,  279,  297,  266],
  [168,  142,  122,  140,  153,  142,  147,  160,  162,  172,  266,  230],
  [298,  277,  346,  263,  253,  239,  268,  228,  240,  260,  252,  241],
  [267,  193,  144,  127,   70,   63,   55,   53,   60,  118,  150,  172],
  [1264, 1234, 1172, 1220, 1547, 1339, 1419, 1453, 1359, 1308, 1264, 1246]);
-> cntab1(nn)?
[3984.0862117, 88.0000000, 0.0000000, 0.0687641, 0.1909168]
```

cntab2

Description:

This function performs a contingency table analysis using entropy measure.

cntab2(nn) ⟶ [h, hx, hy, hygx, hxgy, uxgy uxy]

Where:

nn is an integer matrix representing a two dimensional contingency table,
h is a floating point number representing the entropy of the whole table,
hx is a floating point number representing the entropy of the x distribution,
hy is a floating point number representing the entropy of the y distribution,
hygx is a floating point number representing the entropy of y given x,
hxgy is a floating point number representing the entropy of x given y,
uygx is a floating point number representing the dependence of y on x,
uxgy is a floating point number representing the dependence of x on y,
uxy is a floating point number representing the symmetric dependency.

Pre-variables:

cntab2_tiny [1.0d-30] is a floating point number.

Example:

A contingency table analysis is performed. Note that here we use the same matrix `nn` for input as in `cntab1` above.

```
-> cntab2(nn)?
[4.0368365, 1.5780628, 2.4820040, 2.4587737, 1.5548325, 0.0093595,
0.0147208, 0.0114433]
```

pearsn

Description:

This function computes Pearson's correlation between two data sets.

pearsn(x, y) ⟶ [r, prob, z]

Where:

x is a real list,
y is is a real list having the same length as x,
r is a floating point number representing the correlation coefficient of x and y,
prob is a floating point number representing the significance level at which the null hypothesis of zero correlation is disproved,
z is a floating point number representing Fisher's z statistic.

Example:

The Pearson correlation between two data sets `dose` and `spore` is computed.

```
-> dose : [56.1, 64.1, 70.0, 66.6, 82.0, 91.3, 90.0, 99.7, 115.3, 110.0];
-> spore :[.011, 0.40, 0.37, 0.48, 0.75, 0.66, 0.71, 1.20,  1.01,  0.95];
-> pearsn(dose,spore)?
[0.9013639, 0.0003671, 1.4794450]
```

---

spear

Description:

This function computes Spearman's rank correlation between two data sets.

spear(data1, data2) ⟶ [d, zd, probd, rs, probrs]

Where:

data1 is a real list,
data2 is a real list having the same length as $data1$,
d is a floating point number representing the sum-squared differnce of ranks of the two data sets,
zd is the number of standard deviations by d deviates from its null hypothesis expected value,
probd is a floating point number representing the two sided significance level of zd,
rs is a floating point number representing Spearman's rank correlation,
probrs is a floating point number representing the two sided significance level of the deviation of rs from zero.

Example:

The following table, taken from *Numerical Recipes*, is used as the input data for the functions `spear` and `crank`.

```
-> input : matrix(
[257, 246, 201, 166,  30, 102, 106, 140, 184, 236, 258, 271, 192, 34.0],
[208, 123,  56,  20,   0,   0,   0,  18,  87, 184, 248, 256, 100, 71.0],
[296, 251, 185, 132,  78,  60,  76, 121, 170, 217, 267, 284, 178, 47.0],
[324, 275, 221, 152,  88,  60,  69, 113, 164, 235, 284, 309, 191, 43.5],
[240, 206, 165, 115,  70,  58,  67,  96, 142, 176, 228, 242, 150, 42.5],
[246, 218, 161, 102,  53,  51,  66, 111, 178, 194, 229, 232, 153, 47.0],
[267, 239, 182, 127,  68,  56,  60,  87, 151, 182, 253, 271, 162, 41.5],
[311, 287, 239, 184, 138, 113, 123, 153, 202, 256, 275, 315, 216, 38.0],
[324, 309, 278, 224, 178, 151, 160, 209, 266, 317, 346, 353, 260, 32.0],
[323, 293, 243, 182, 117,  77,  90, 143, 212, 264, 308, 337, 216, 37.0],
[263, 235, 197, 156, 118,  95,  97, 134, 171, 227, 257, 273, 185, 36.0],
[305, 293, 271, 245, 208, 176, 175, 200, 234, 262, 300, 297, 247, 21.0],
[270, 250, 214, 167, 118,  91,  96, 127, 173, 220, 256, 272, 188, 35.0],
[260, 246, 216, 188, 171, 154, 166, 201, 238, 263, 267, 257, 219, 26.0],
[251, 238, 175, 127,  77,  62,  71, 102, 151, 183, 220, 255, 159, 41.0],
[275, 252, 192, 142,  96,  80,  99, 134, 172, 224, 248, 272, 182, 21.0],
[288, 262, 208, 152,  99,  76,  90, 135, 193, 235, 259, 287, 190, 44.0],
[242, 209, 150,  84,  44,  29,  34,  60, 118, 174, 216, 228, 132, 47.5],
[304, 286, 281, 216, 172, 144, 151, 195, 264, 322, 358, 343, 253, 41.0],
[267, 190, 196, 145,  75,  64, 101, 124, 153, 182, 215, 247, 163, 39.0]);


-> july : column(1,input)?
[257, 208, 296, 324, 240, 246, 267, 311, 324, 323, 263, 305, 270, 260, 251,
275, 288, 242, 304, 267]
-> august : column(2,input)?
[246, 123, 251, 275, 206, 218, 239, 287, 309, 293, 235, 293, 250, 246, 238,
252, 262, 209, 286, 190]
-> spear(july,august)?
[122.0000000, -3.9584579, 0.0000754, 0.9081325, 0.0000000]
-> november : column(5,input)?
[30, 0, 78, 88, 70, 53, 68, 138, 178, 117, 118, 208, 118, 171, 77, 96,
 99, 44, 172, 75]
-> spear(july,november)?
[390.5000000, -3.0776421, 0.0020865, 0.7060595, 0.0005034]
```

crank

Description:

This function replaces array elements by their ranks.

$$\mathbf{crank(w)} \longrightarrow \mathbf{[w,\ s]}$$

Where:

w on input is a sorted real list and on output a list of the ranks,
s is a floating point number representing the sum of $f^3 - f$ where f is the number of elements in each tie.

Example:

Here we use the july column from the matrix used in `spear` above.

```
-> order : [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20];
-> res : sort2(july,order)?
[[208.0, 240.0, 242.0, 246.0, 251.0, 257.0, 260.0, 263.0, 267.0, 267.0, 270.0,
275.0, 288.0, 296.0, 304.0, 305.0, 311.0, 323.0, 324.0, 324.0], [2.0, 5.0,
18.0, 6.0, 15.0, 1.0, 14.0, 11.0, 7.0, 20.0, 13.0, 16.0, 17.0, 3.0, 19.0, 12.0,
8.0, 10.0, 9.0, 4.0]]
-> rank : crank(first(res))?
[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.5, 9.5, 11.0, 12.0, 13.0, 14.0,
15.0, 16.0, 17.0, 18.0, 19.5, 19.5], 12.0]
```

kendl1

Description:

This funtion computes Kendall's tau for the correlation between two data sets.

kendl1(data1, data2) $\longrightarrow$ **[tau, z, prob]**

Where:

data1 is a real list,
data2 is a real list having the same length as $data1$,
tau is a floating point number representing Kendall's τ,
z is a floating point number representing the number of standard deviations from z of τ,
prob is a floating point number prepresenting the two sided significance of τ.

Example:

Kendall's τ for the correlation of a data set with itself and of two data sets is computed.

```
-> data1 : [1,2,3,4,5,6];
-> kendl1(data1,data1)?
[1.0000000, 2.8180093, 0.0048323]
-> data2 : [6,5,4,3,2,1];
-> kendl1(data1,data2)?
[-1.0000000, -2.8180093, 0.0048323]
```

kendl2

Description:

This function performs a contingency table analysis using Kendal's tau.

kendl2(tab) ⟶ [tau, z, prob]

Where:

tab is a real matrix with entries $tab_{k,l}$ such that $tab_{k,l}$ is the number of events falling in bin k of one variable and bin l of another,
tau is a floating point number representing Kendall's τ,
z is a floating point number representing the number of standard deviations from z of tau,
prob is a floating point number prepresenting the two sided significance of tau.

Example:

A contingency table analysis for a 5 by 20 table is performed using `kendl2`.

```
-> Random numbers from Etons Tables: #
-> tab : matrix([2,3,8,7,9,0,0,3,3,9,5,1,5,6,9,5,1,2,8,4],
                [7,3,5,9,1,2,5,0,7,0,3,6,2,9,1,6,7,5,6,2],
                [3,2,0,0,2,8,7,6,4,5,9,1,6,6,9,5,0,5,7,4],
                [3,1,0,0,0,5,8,6,5,8,3,3,5,5,9,3,7,1,8,1],
                [8,6,6,1,0,5,8,4,4,7,3,6,6,8,3,4,6,1,8,9]);
-> kendl2(tab)?
[0.0259665, 0.8221251, 0.4110060]
```

smooft

Description:

This function smooths a data set using the fast fourier transform.

smooft(y, pts) ⟶ y

Where:

y on input is a real list and on output the corresponding smoothed list,
pts is a real number representing the width of the window of neighbouring points used for smoothing.

Pre-variables:

smooft_mmax [1024] an integer being the maximum size of a padded array.

Example:

A data set of 17 elements is smoothed using `smooft`.

```
->  y : [1,2,3,4,5,6,7,8,9,8,7,6,5,4,3,2,1];
-> smooft(y,3)?
[1.2662296, 1.9893345, 2.9822691, 4.0133558, 5.0026346, 5.9788796, 7.0273666,
8.0253642, 8.4684184, 8.0253642, 7.0273666, 5.9788796, 5.0026346, 4.0133558,
2.9822691, 1.9893345, 1.2662296, -0.0144925, -0.0117148, 0.0095345, -0.0026346,
-0.0017698, 0.0020792, -0.0002063, -0.0008776, -0.0002063, 0.0020792,
-0.0017698, -0.0026346, 0.0095345, -0.0117148, -0.0144925]
```