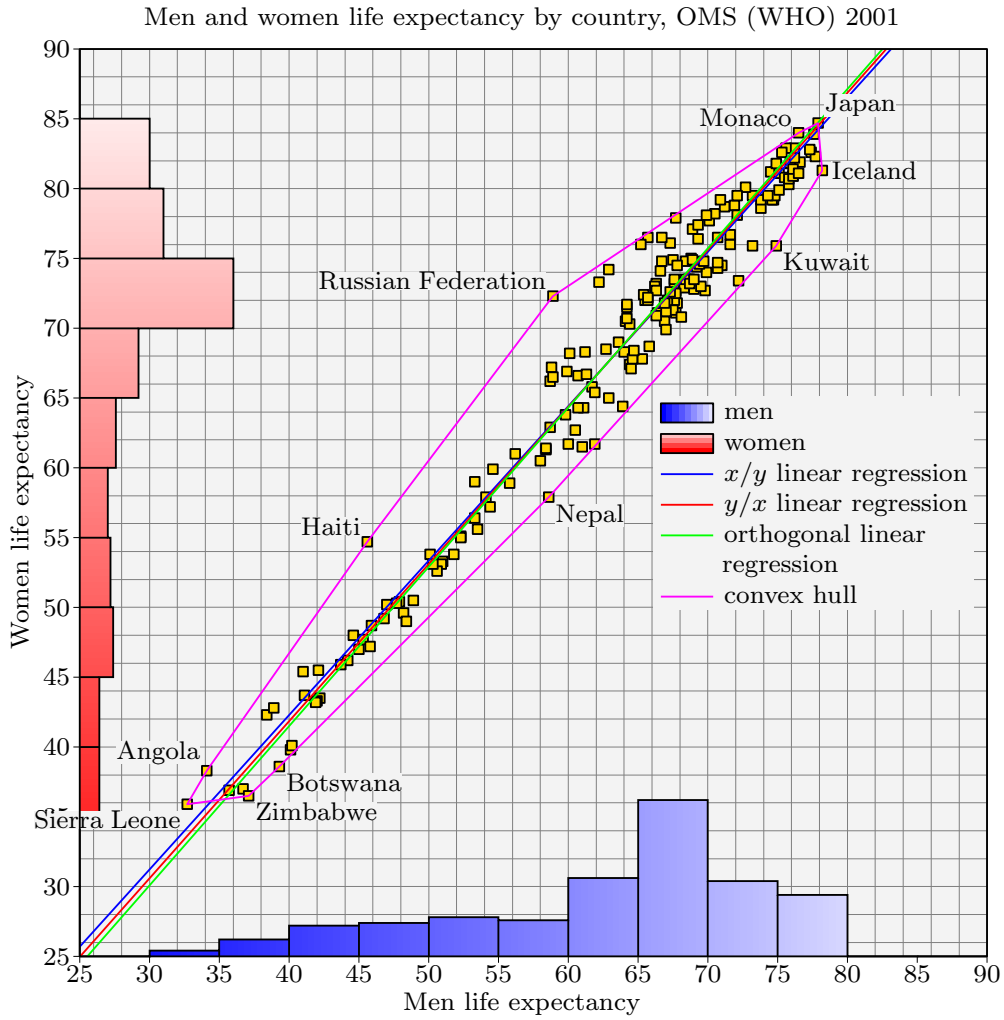
METAPOST FOR PROBABILITY AND STATISTICS

MetaPost Promotion Pages

by Anthony Phan



Prerequisites

To use the `mps` suite (`mps-core.mp`, `mps-math.mp`, `mps-stat.mp`), one has to be quite acquainted with the MetaPost language, that is one has to have read the *MetaPost manual* and have used this language to draw pictures. A second requirement is of course knowing elementary statistics.

This text is just an incomplete draft of a manual of the unstable and incomplete package `mps`. Consider this just as examples of MetaPost use.

Syntax, algorithms, and such, are just coming from the way I think about it, and of course also of what I have experienced with some other computer programs. All of this does not have a very ambitious aim. Mine is just to draw nice diagrams for some use in pedagogical notes.

§ 1. Data input and use

Data input. — This program does not aim to manage large sets of data, or even sets of large data, there are other programs for that. One of the limitation is the size of the input stack since data are simply considered as lists read by loops. Another limitation is that one cannot access to some $x[i]$ when $i \geq 4096$.

A data set is a list of numerics or/and strings that can be handled by MetaPost capabilities (also booleans, pairs, colors, cmykcolors). Defining a data set is just:

`data(`⟨*dataname*⟩`)` ⟨*value*⟩$_1$`,` ⟨*value*⟩$_2$`, ...;`

which defines a hidden control sequence (`_data_`⟨*dataname*⟩) whose replacement text is just the list of values.

Data use. — A data set is a set of entries of one, or more, variable. How to use a defined data set depends on its content. The general syntax is

`usedata(`⟨*dataname*⟩`,` ⟨*variable*⟩$_1$`,` ⟨*variable*⟩$_2$`, ...);`

where ⟨*variable*⟩$_i$ is a sequence of characters which defines the name of the corresponding variable. If ⟨*variable*⟩$_i$ does not end with `$` nor $0, 2, 3, 4$ then a column of numeric variables with name ⟨*variable*⟩$_i$[.] will be created destroying any former meaning of ⟨*variable*⟩$_i$. If ⟨*variable*⟩$_i$ does end with `$`, say ⟨*variable*⟩$_i$ = ⟨*variable*⟩$'_i$`$`, then a column of string variables with name ⟨*variable*⟩$'_i$[.] will be created destroying any former meaning of ⟨*variable*⟩$'_i$. In fact, all possible type tags are the following:

- ⟨*variable*⟩`$` declares a string variable;
- ⟨*variable*⟩`0` declares a boolean variable;
- ⟨*variable*⟩`1` declares a numeric variable (very optional);
- ⟨*variable*⟩`2` declares a pair variable;
- ⟨*variable*⟩`3` declares a color (triple) variable;
- ⟨*variable*⟩`4` declares a cmykcolor (quadruple) variable.

Then every variable receives a value according to the values' list in the data set. That also means that the number of values should be a multiple of the number of variables and that their types should be coherent to the values' list.

Moreover, if a variable is named ⟨*variable*⟩, then two numerics are set: ⟨*variable*⟩`.n` which is the number of entries of this variable, and ⟨*variable*⟩`.sortflag` whose value is 0 since no sorting has been done. A new string named ⟨*variable*⟩`.type` is set to `"numeric"`, `"string"`, `"boolean"`, `"pair"`, `"color"`, `"cmykcolor"`, according to the type of the ⟨*variable*⟩.

Important remark. — In fact, a boolean named `declareflag` may be set to `true` in order to check if any new ⟨*variable*⟩ declared in `usedata` has a former meaning. What will be tested is if `known` ⟨*variable*⟩ is true or not. If ⟨*variable*⟩ is totally unknown, nothing would happen; if ⟨*variable*⟩ is a known primary, a warning message should be issued; if ⟨*variable*⟩ is the name of some control sequence with arguments, an error should happen.

If something mysteriously goes wrong within a program, it may be nice to check whether this can be due or not to the names of some new variables or control sequences. The boolean `declareflag` provides a little help for that. Also, even if `declareflag` = `false`, errors may occur rather immediately if ⟨*variable*⟩ is the name of some control sequence with arguments.

Ignoring variables' entries. — Using a data set, all variables entries are not necessarily needed, so assigning them should be avoided. For this purpose there is a special reserved variable's name which is `drop`, it means that when a variable name in the variables' list is `drop` then no new variable is created and the corresponding entries are ignored.

Conditional use of data. — The preceding subsection shows how to ignore some columns in a data set. Ignoring rows in a data set is done with

`usedataif` ⟨*condition*⟩`; usedata(`⟨*dataname*⟩`,` ⟨*variable*⟩$_1$`,` ⟨*variable*⟩$_2$`, ...);`

The ⟨*condition*⟩ must deal with declared variables (in `usedata`). These variables must appear in the ⟨*condition*⟩ as ⟨*variable*⟩$_i$`@#`. The suffix `@#` is the index of the current row.

First example. — Here, the data set is a simple list of numerics. The `usedata` control sequence uses the dataset "Eggs" and sets `x[1] = 688`, ..., `x[20] = 557`, `x.n = 20` and `x.sortflag = 0`.

```
data(Eggs)
  688, 388, 544, 645, 264, 322, 503, 847, 540,
  382, 404, 203, 422, 307, 784, 402, 556, 419, 690, 557;

beginfig(1);
  ...
  usedata(Eggs, x);
  ...
endfig;
```

Second example. — Here, the data set is more complex. The `usedata` control sequence uses the dataset "Spiders" and sets:

- `sex[1] = "female"`, ..., `sex[40] = "male"`, `sex.n = 40`, `sex.sortflag = 0`;
- `size[1] = 1.49`, ..., `size[40] = 0.28`, `size.n = 40`, `size.sortflag = 0`;
- `eggs[1] = 688`, ..., `eggs[40] = 0`, `eggs.n = 40`, `eggs.sortflag = 0`;

```
data(Spiders)
  "female", 1.49, 688, "female", 1.17, 388, "female", 1.29, 544,
  "female", 1.36, 645, "female", 1.05, 264, "female", 1.11, 322,
  "female", 1.26, 503, "female", 1.59, 847, "female", 1.29, 540,
  "female", 1.17, 382, "female", 1.18, 404, "female", 0.88, 203,
  "female", 1.20, 422, "female", 1.10, 307, "female", 1.44, 784,
  "female", 1.18, 402, "female", 1.30, 556, "female", 1.20, 419,
  "female", 1.39, 690, "female", 1.30, 557,
  "male", 0.37, 0, "male", 0.40, 0, "male", 0.44, 0, "male", 0.58, 0,
  "male", 0.42, 0, "male", 0.37, 0, "male", 0.40, 0, "male", 0.38, 0,
  "male", 0.29, 0, "male", 0.47, 0, "male", 0.51, 0, "male", 0.43, 0,
  "male", 0.50, 0, "male", 0.37, 0, "male", 0.39, 0, "male", 0.51, 0,
  "male", 0.62, 0, "male", 0.48, 0, "male", 0.50, 0, "male", 0.28, 0;

beginfig(1);
  ...
  usedata(Spiders, sex$, size, eggs);
  ...
endfig;
```

If `sex` doesn't matter, just type

```
usedata(Spiders, drop, size, eggs);
```

if `eggs` doesn't matter just type

```
usedata(Spiders, sex$, size, drop);
```

if `sex` and `eggs` don't matter just type

```
usedata(Spiders, drop, size, drop);
```

Third example. — In the data set "Spiders", one may want to use only data related to females in order to study the number of eggs for instance. This can be done as the following:

```
beginfig(1);
  ...
  usedataif sex@# = "female";
  usedata(Spiders, sex$, drop, eggs);
  ...
endfig;
```

or in this peculiar case:

```
beginfig(1);
  ...
  usedataif eggs@# > 0;
  usedata(Spiders, drop, drop, eggs);
  ...
endfig;
```

Declaring data. — One may want to use data that doesn't come from a `data` statement. For instance when one wants to generate random outcomes:

```
save x;
numeric x[];
for i = 1 upto 100:
  x[i] = uniformdeviate(1);
endfor
```

Then one has to set `x.n` to its value: here,

```
x.n = 100;
```

It is not necessary to set `x.sortflag` since an unknown value of it is equivalent to the value 0, meaning that the corresponding data are not sorted.

A better way is to declare the variable and the number of its entries:

```
declare(x, 100);
for i = 1 upto x.n:
  x[i] = uniformdeviate(1);
endfor
```

In this case, if `declareflag` = `true`, it is checked if `x` has a former meaning resulting to a warning message or an error; then—if no error occurs—, `x` is saved, defined as a numerical row, `x.n` is set to 100, `x.sortflag` is set to 0, `x.type` is set to `"numeric"`.

The `declare` control sequence is similar to `usedata`: `declare(sex$, 100)` would make, if no error occurs, `sex` saved, defined as a string row, `sex.n` set to 100, `sex.sortflag` set to 0, `sex.type` set to `"string"`; `declare(point2, 100)` would make, if no error occurs, `point` saved, defined as a pair row, `point.n` set to 100, `point.sortflag` set to 0, `point.type` set to `"pair"`, etc.

Sorting data. — There are many reasons for sorting data. It is rather easy to do:

```
sort x;
```

The numeric `x.sortflag` is then set to 1. One can sort many variables' entries simultaneously:

```
sort x, y, z;
```

The sorting procedure will be done on every `x.n` first entries of the variables `x`, `y` and `z`. Thus, the numbers of entries of should match. In fact, what is done is the usual lexicographical sort of the (sub)array specified by the list of variables. In this example, `x.sortflag` is set to 1, and `y.sortflag` and `z.sortflag` are set to 0.

The sort algorithm used here is the so-called "heap sort" algorithm.

Sort is usually done with the ascending order, but sometimes, it one may prefer to do it in descending order. The internal numeric `sorttype` has been introduced for that. When it has a strictly positive value (say 1), the sort is done with ascending order; otherwise it is done with descending order.

Weightening data. — It is quite usual that data may be weightened, the weight variable meaning that some entries are repeated accordingly. For example,

```
data(poll)
  "pros", 26,
  "cons", 32,
  "no opinion", 17;

usedata(poll, opinion$, n);
```

Then one has to declare that the variable `opinion` is weightened by the variable `n`. It is done with

```
weight opinion, n;
```

Then, the string `opinion.w` is defined to be `"n"`, and the numeric `opinion.tw` is set to `n[1]` + ⋯ + `n[n.n]`, that is the total weight for the variable `opinion`. One should notice that use of `weight` as a variable name should be forbidden. In fact the general syntax of the `weight` statement is

`weight` ⟨*variable*⟩$_1$`,` `...,` ⟨*variable*⟩$_k$`,` ⟨*weight-variable*⟩`;`

which assigns to every ⟨*variable*⟩$_i$ the weight ⟨*weight-variable*⟩ as described for the variable `opinion` in the previous example. The need for weightening variables lies in statistical computations like means, frequency diagrams, ... The weight variable should be made of positive numerics, not necessarily integers, which does not have to sum up to one or whatsoever. The total weight is simply stored in ⟨*weight-variable*⟩`.tw` and also in ⟨*variable*⟩`.tw` just as

the string ⟨*variable*⟩`.w` is defined to be the ⟨*weight-variable*⟩'s name. It is also possible to unweight a variable or a list of variables with

`unweight` ⟨*variable*⟩$_1$`, ...,` ⟨*variable*⟩$_k$`;`

What would remain of each variable are just their entries, their number and their sortflag. (What is done is copying each variable into a temporary one, then copy it back; *see* the next subsection.)

Copying variables. — This may be helpful to copy a variable x into a new variable y, for instance when one wants to make some special computations or reordering without destroying the former structure of x, or to extract only some entries of the variable x. The general syntax to do so is

`copy(`⟨*known-variable*⟩$_1$`,` ⟨*new-variable*⟩$_1$`,...,`⟨*known-variable*⟩$_k$`,` ⟨*new-variable*⟩$_k$`);`

Here variables names go by pairs and there can be one too many such pairs. Copying can also be conditional

`copyif` ⟨*condition*⟩`;` `copy(`⟨*known-variable*⟩$_1$`,` ⟨*new-variable*⟩$_1$`,...);`

The `copyif` statement has exactly the same meaning as `usedataif` and also the same syntax. But there is no need for the ⟨*condition*⟩ to involve, only, variables to be copied. The new variables get the `type` of the original ones, the `sortflag` of them if they are known or else 0, and the `n` (total number of entries) corresponding to the condition. The copy is done pair by pair. This means that the various known variables need not to have the same number of entries. For example, try:

```
usedata(Spiders, sex$, size, eggs);
copyif (sex@# = "female") and (size@# > 1.3);
copy(sixe, x, eggs, y);
sort x, y;
for i = 1 upto x.n: message decimal x & ", " & decimal y; endfor
```

If the new variables are to be weightened, it is better to copy the weight entries within the same `copy` statement, especially if it is a conditional copying; since new variables are not weightened, it would be necessary to use the `weight` statement once again. For instance,

```
data(poll)
  "pros", 26,
  "cons", 32,
  "no opinion", 17;

usedata(poll, opinion$, n);
weight opinion, n;
copyif opinion@# <> "no opinion";
copy(opinion, op, n, opn);
weight op, opn;
```

§ 2. More about data management

Reading data from a file. — There are plenty of data file formats. We deal only with the rawest ones: more or less `.dat` or `.csv`-files. A data file is a text file. It is made of lines of text. Each line contains strings and/or numerics which could be handled by MetaPost.

A single data is defined by what precedes it and what follows. A beginning of line or/and delimiting character(s) defines the beginning of a new entry. An end of line or/and delimiting characters defines the end of an entry. The delimiting character is defined by a MetaPost string named `DLM`. Its default value is `DLM := " "`, that is, a normal space. One can set it to `DLM := TAB` where `TAB` is predefined to be the tabulation character. Other cases like `DLM := ","` may appear (*comma separated volume* or `csv`). The exact pattern recognition scheme is the following

⟨*beginning of the line or DLM*⟩⟨*some spaces*⟩⟨*actual entry*⟩
⟨*some spaces*⟩⟨*end of line or DLM*⟩
⟨*beginning of the line or DLM*⟩⟨*some spaces*⟩`QUOT`⟨*actual entry*⟩`QUOT`
⟨*some spaces*⟩⟨*end of line or DLM*⟩

Beware, `QUOT` in the previous line can be almost any single character matching the value of `SQUOT` (default value ') or `DQUOT` (default value "). Entries beginning or ending with spaces—or made of spaces, even none—must be surrounded by quotes.

An entry would be recognized as a numeric if it is read as a numeric. Otherwise, it would be recognized as a string. An entry would be systematically recognized as a string if it is surrounded by, respectively `SQUOT` (single quote) or `DQUOT` (double quot). These are MetaPost string variables whose default values are respectively single (') and double quotes ("). They may be changed, but we don't think that it would be a good idea.

Also, expressions like `6.62e-34` or `1.51+i0.31` are read as strings.

Missing, or more exactly blank, values are ignored. This means that many delimiting characters or ends of line would be treated as a single delimiter. So, missing values must be coded with some keywords in the data file. The keyword used to replace missing numeric and string entries in data files is defined as the value of the MetaPost string `NA`. Its default value is

```
NA := "NA"
```

but one can set `NA := "."` (Sas convention), or `NA := "undefined"` (Maple convention), or even `NA := "?"` (which looks fine).

Some lines maybe treated as comments. This is the case when their *very first* characters form a string matching predefined MetaPost string `COMMENT`. Its default value is `"#"` and maybe changed (for example, `COMMENT := "%"`, or `COMMENT := "comment"`, or anything else).

Reading a data file is done with

`readdata(`⟨*datafile*⟩`,` ⟨*dataname*⟩`)`

where ⟨*datafile*⟩ is a string (string variable or explicit name surrounded by double quotes) and ⟨*dataname*⟩ the name of the data set where the contents of the data file would be stored as a simple MetaPost list (see former sections). In such list, what would have been recognized as numerics would be written as usual MetaPost numerics, and what would have been recognized as strings would be written as usual MetaPost strings. Missing values `NA` would be stored as the MetaPost variable `NA` and not its current value (this means that the value can be changed even if the dataset is stored into MetaPost memory).

One can read a data file from one line to another: it suffices to set MetaPost numerics `firstdataline` and `lastdataline` to the corresponding values before reading the data file. After the reading, their values would be restored to their default ones:

```
firstdataline := 0; lastdataline := infinity;
```

Finally, the data could then be accessed by a `usedata(`⟨*dataname*⟩`, ...)` invocation.

Example. — The following lines produce exactly the same `oms` data set and variables.

```
DLM := " "; COMMENT := "#"; readdata("oms.dat", oms);
usedata(oms, country$, x, y);

DLM := ","; COMMENT := "%"; readdata("oms.csv", oms);
usedata(oms, country$, x, y);
```

But, beware. In `oms.csv` entries are separated by and only by `","` but it also works with some `", "`. Also commas before ends of lines could have been removed and `"%"` as a comment character may be not standard.

Storing variables' entries into a data set. — One can store the entries of declared variables into a data set. These variables can be numerics, pairs, or strings. It suffices to type

`storedata(`⟨*dataname*⟩`,` ⟨*variable*⟩$_1$`,` ⟨*variable*⟩$_2$`, ...);`

If the numbers of entries of the variables differ, or if there are unknown values, the value of `NA` would be used to fill the gaps (so these gaps are filled with a predefined string even if a numeric should be there). If the number of entries ⟨*variable*⟩`.n` of at least one variable is undefined, storage is aborted. If the storage can be done, a track of the number of variables involved is kept into some hidden numeric for future use.

The storage can be conditional: just use `storedataif` (it is the same thing as `usedataif` and `copyif`) before running `storedata`.

Writing a data set into a file. — To write a data set into a new data file, just type:

`writedata(`⟨*dataname*⟩`,` ⟨*datafile*⟩`)`

If there is some track of the number of variables involved into the data set (*see* the previous subsection), each line of the data file would correspond to one entry of each variable; if not, there would be just one entry on each line. Except that, the data file would follow the same conventions as the ones described in the subsection about reading data from a file: delimiters `DLM` and/or ends of lines, strings surrounded by `DQUOT`, missing values coded as the value of `NA`, and finally numerics, pairs, colors, cmykcolors are written with a control sequence ⟨*type*⟩`tostring` (`pairtostring` for instance default behavior produces things like `6.62e-34` but can be easily be turned to something else).

If one needs to add headers or comments at the beginning of the data file, one can type:

```
dataheader ⟨list of strings⟩;
writedata(⟨dataname⟩, ⟨datafile⟩);
dataheader;
```

The first call of `dataheader` defines the lines of comments at the beginning of the ⟨*datafile*⟩, the last call empties the list (but one can still keep it for further use).

Conditional use of data with NA (don't trust what follows). — Dealing with missing data is rather painful. One may use or copy data if and only if the corresponding entries are defined. In `mps-core.mp`, one can find the following definition:

```
def notNA primary x =
  if known x:
    if string x:
      if x = NA: false else: true fi
    else: true fi
  else: false fi
enddef;
```

This provides a simple and–I hope–clear way to use or copy data conditionally. Here is an example:

```
copyif (espece@# = "ACRSCI") and (sexe@# = "F") and (notNA longueur@#);
copy(longueur, x);
```

The database deals with birds. Species and sex data are fully available, but length is not. Of course, there should be more complex situations where for instance species could be NA. In this case, I suppose that

```
copyif if known espece@#: espece@# = "ACRSCI" else: false fi;
```

In this later example, the variable "espece" has already been defined by some `usedata` statement. Missing values have been left undefined. The situation would be different if the variable has to be defined directly from the database.

```
usedataif if notNA espece@#: espece@# = "ACRSCI" else: false fi;
```

```
usedata(oiseaux,  NF, espece$,
  sexe$, age$, adiposite, adkleins, longueur,
  masse, PC, PI, TL$, CS$);
```

Remark. — This paragraph has to be checked carefully... A problem that remains is to use data only when one of the variables is undefined. But there is no problem to copy data with a condition on a variable to be undefined... Further more, when an entry has to be rejected, just do `x[21] := undefined` for instance. Then, if you copy things you just have to require that `notNA x@#`.

Little bonus. — We have add a few control sequences into `mps-core` for reading data files and converting them into R, Maple, Scilab or Matlab declarations. Here is an example.

```
input mps-core;
readdata("atlas.dat", donnees);
usedata(donnees, country$, region$, x, y, surf$, nhab$);
stringtofloat(nhab, habitants); stringtofloat(surf, superficie);
export.scilab("atlas.sce", country, region, x, y, superficie, habitants);
end.
```

In `atlas.dat` the two last columns provide numbers with scientific notation. Thus they are read as lists of strings and stored into `donnees`. We use them as string variables which are converted to pairs by the control sequence `stringtofloat`. The control sequence `export.scilab` just write the imperative declarations of each variables into a new file `atlas.sce`. In fact the true control sequence is `export.@#` and suffixes like `scilab`, `matlab`, `R` and `maple` are recognized and undefined values are replaced by some convenient keyword specific to each case. Other would lead to understandable output anyway.

Line width in the output is controlled by the numeric `export_line_width` whose default value is 60 but can be set to 0 if one wants only one entry per line for instance.

Classification. — Here we deal with one or many variables with the same number of entries n. We want to produce some classification, or ranking, without changing any of the previous variables. Thus a new variable named `index` will be created to contain the information.

`bounds` ⟨*text*⟩. Control sequence. It defines `currentbounds` as the text given as argument. The first item is also stored in `_lb_` and the last in `_ub_` for some practical reasons. So the text must be an increasing sequence of numerical values.

`classification` ⟨*closure*⟩. Control sequence. Suppose that `currentbounds` is an increasing list $a_0, \ldots, a_k$. With `classification` ⟨*right*⟩, the corresponding classes will be $[a_0, a_1]$, $]a_1, a_2]$, ..., $]a_{k-1}, a_k]$. With `classification` ⟨*left*⟩, the corresponding classes will be $[a_0, a_1[$, $[a_1, a_2[$, ..., $[a_{k-1}, a_k]$. Default setting is `classification` ⟨*left*⟩.

`classify` ⟨x⟩. Control sequence. From 1 to `x.n`, it sets `index[i]` to the middle of the class to which `x[i]` belongs, the classes being defined by `currentbounds`.

`numberize` ⟨x⟩. Control sequence. If there is k different values, than every value of ⟨x⟩...

`rank` ⟨x⟩. Control sequence. Provides a rank for every value of ⟨x⟩, i.e. a number between 1 and `x.n`...

`kmeans(`⟨k⟩`)` ⟨*text*⟩. Control sequence.

§ 3. Basic computations

`means` ⟨*undelimited text*⟩. Control sequence. It computes the mean of every numeric ⟨*variable*⟩ named in ⟨*undelimited text*⟩ and stores the result in ⟨*variable*⟩`.mean`. If some variable is not of numeric type, an error should occur. This computation is not very accurate, it does not sum up the, eventually weightened, values then divide by the total weight, but performs successive means computation in order to keep numerical values in the range of MetaPost capabilities. The three quartiles are computed for a variable in ⟨*undelimited text*⟩ if and only if its entries are sorted (⟨*variable*⟩`.sortflag = 1`) (*see* the quantile function), and then stores them in ⟨*variable*⟩`.q1`, ⟨*variable*⟩`.q2`, ⟨*variable*⟩`.q3`. Minimum and maximum of entries need not to be computed if the entries are sorted since in this case they are respectively ⟨*variable*⟩`[1]` and ⟨*variable*⟩`[`⟨*variable*⟩`.n]`.

`quantile` ⟨*delimited pair*⟩. Control sequence. The function quantile has for arguments a delimited pair `(`⟨*variable*⟩`,` α`)`, where $\alpha \in [0, 1]$, and returns the quantile of order α of the sample ditribution of ⟨*variable*⟩. It assumes that the entries of ⟨*variable*⟩ are sorted and just gives a warning if ⟨*variable*⟩`.sortflag` is not equal to 1. Computations are done in the usual way: when α faces a jump of the cumulative function of the discrete distribution of ⟨*variable*⟩, it returns the value of the variable where the jump occurs; when α faces a proper step of the cumulative function, it returns the middle of the interval over which this step lies; when $\alpha = 0$ or 1, it returns respectively the minimum or maximum value of ⟨*variable*⟩.

`variances` ⟨*undelimited text*⟩. Control sequence. It computes the (population) variance, (population) standard deviation, (sample) corrected variance, (sample) corrected standard deviation, of every, eventually weightened, variable named in ⟨*undelimited text*⟩ and stores the results in, respectively, ⟨*variable*⟩`.var`, ⟨*variable*⟩`.std`, ⟨*variable*⟩`.cvar`, ⟨*variable*⟩`.cstd`. If ⟨*variable*⟩`.mean` is unknown then it is computed with the `means` function. If some variable is not of numeric type, an error should occur. This computation is still not very accurate.

`covariances` ⟨*undelimited text*⟩`.` Control sequence. It makes sure that every ⟨*variable*⟩ in ⟨*undelimited text*⟩ has a known variance ⟨*variable*⟩`.var` and then a known mean ⟨*variable*⟩`.mean`; if not, computations are done according to the former control sequences. Covariances are computed with the same lack of accuracy as before, they are stored in numeric variables ⟨*variable*⟩$_i$`.`⟨*variable*⟩$_j$`.cov` and ⟨*variable*⟩$_j$`.`⟨*variable*⟩$_i$`.cov` which are equal. There is no ⟨*variable*⟩$_i$`.`⟨*variable*⟩$_i$`.cov` since it should be equal to ⟨*variable*⟩$_i$`.var`. If ⟨*variable*⟩$_i$ and ⟨*variable*⟩$_j$ are incompatible (different number of entries or different weightening) their covariance is set to 0 since there should be no correlation between them.

Remark. — In `means` or `variances` variables need not to be weightened the same way, or to have the same number of entries.

§ 4. Setting up graphics and coordinate system

For all kind of graphics (even `piechart`), a scale must be given. This is done with

`setrange(`$x_{\min}$`,` $x_{\max}$`,` ⟨*width*⟩`,` $y_{\min}$`,` $y_{\max}$`,` ⟨*height*⟩`);`

then `gcoord(`x`,` y`)`, ...

Predefined pens. — There are three predefined pens which are `thin.nib`, `rule.nib` and `thick.nib` and which are "pencircles" respectively scaled by the following dimensions:

$$\texttt{thin} = 0.2\,\texttt{pt}, \qquad \texttt{rth} = 0.4\,\texttt{pt}, \qquad \texttt{thick} = 0.6\,\texttt{pt}.$$

One would recognize the dimension `rth` to be the usual "rule thickness" of TEX—or, more precisely, of 10 points size Computer Modern fonts. These three pens don't really appear into the graphical procedures. In fact, there are two cases: use of the `currentpen` for drawing graphs or contours, use of special dimensions for special tasks. These special dimensions are

`tickbreadth`, `gridbreath`, `hatchbreath` and `framebreadth`.

They are related to pens named `tick.nib`, `grid.nib`, `hatch.nib` and `frame.nib`. The basic values of these dimensions are set to be one of the three former dimensions.

Applying changes of dimensions (`thin`, `rth`, `thick`, `gradbreath`, `gridbreath`, `hatchbreath` and `framebreath`) to the corresponding pens can be done by executing the `pens_setup` control sequence. For instance:

```
thin := 0.18pt; rth := 0.3pt; thick := 0.5pt;
gradbreath := gridbreadth := hatchbreath := thin; framebreath:= thick;
pens_setup;
```

§ 5. Graphical control sequences and parameters

`gcoord` ⟨*pair of relative coordinates*⟩`.` Control sequence. It returns the absolute coordinates corresponding to the relative coordinates taking also care of offsets. It stands for "planar coordinates" and may be changed for special tasks.

`setscoord(`⟨*expression*⟩(x, y)`,` ⟨*expression*⟩(x, y)`).` Control sequence. It defines the `scoord` control sequence explained thereafter.

`scoord` ⟨*pair of relative coordinates*⟩`.` Control sequence. It returns the relative coordinates defined by the formulas in `setscoord`. The relative coordinates of special (statistical) graphics should be defined through this control sequence. Then, it would be very easy to move or transform such graphics (*in progress*).

`gdraw` ⟨*path in relative coordinates*⟩`.` Control sequence. It just draws its argument with respect to current coordinates and offsets and also with pen `currentpen`, color `graphcolor`, options set by `gdrawoptions(...)` and specific options given after the path. It is used in most of graphical procedures.

`graphcolor.` Color parameter. This parameter is systematically used when calling the `gdraw` control sequence. It can be changed and at time, and one can often bypass its value with options like `withcolor` ⟨*some color*⟩.

`gdrawoptions(`*text*`).` Control sequence. It has exactly the same use and role as plain-MetaPost `drawoptions`. One can use it for changing the default appearance of basic diagrams.

`gdot` ⟨*pair of relative coordinates*⟩`.` Control sequence. It just draws the `gdotpath` scaled by `dotlabeldiam` at the relative coordinates given as argument. Options are `gdraw` options. But one can use it as

`gdot(`x`,`y`) withcolor` ⟨*somecolor*⟩

in order to superset the default color which is `graphcolor`.

`gdotpath.` Path parameter. Can be user defined.

`gdotstyle(`x`, `y`).` Control sequence. The argument x resets the `gdotpath`: 0 fullcircle, 1 square, 2 diamond, 3 triangleup, 4 triangledown, 5 pentagon, 6 hexagon. The argument y sets the drawing mode: 0 for drawing contour with defaultcolor `graphcolor`, 1 for filling contour with defaultcolor `graphcolor`, 2 for filling contour with color `fillcolor` and drawing contour with defaultcolor `graphcolor`.

`gdotdraw` ⟨*number*⟩`.` Control sequence. Default meaning is `fill` but may be changed to `draw` for instance.

`title[.`⟨*location*⟩`](`⟨*text*⟩`).` Control sequence. It draws the list of pictures (defined through `btex ... etex` for instance) and strings given in argument from left to right. Numerical arguments appearing strictly before the last argument are printed, but a last numerical argument would be read as an angle (in degrees) for rotating the title. The resulting picture is placed at one of the eight possible locations: `top`, `urt`, `rt`, `lrt`, `bot`, `llft`, `lft`, `ulft`. This is done with respect to the range given at the beginning and also items that would overpass this range (annotations, graduations, ...) in order to label the whole figure. Such a control sequence should be used at the very end of each figure. It uses the `textcolor` parameter, so its output is always single colored.

`titleoffset.` Numeric parameter. This dimension is one the parameters of `title` (remember also `textcolor`). Its predefined value is 6 pt and its meaning should be obvious.

`xticks[.`⟨*location*⟩`](`⟨*relative vertical location*⟩`)` ⟨*list of relative horizontal coordinates*⟩`.` It uses the `textcolor` parameter so its output is always single colored.

`yticks[.`⟨*location*⟩`](`⟨*relative horizontal location*⟩`)` ⟨*list of relative vertical coordinates*⟩`.` It uses the `textcolor` parameter so its output is always single colored.

`tickbreadth.` Numeric parameter.

`ticklength`. Numeric parameter, its default value is 0.6666`labeloffset`;

`tickoffset`. Numeric parameter, its default value is `labeloffset` − `ticklength`;

`autograd[.`⟨*location*⟩`]`. Automatically grades the axis. One can grade only one axis with `autograd.bot` or `autograd.lft`, or both with `autograd`, `autograd.llft`, `autograd.urt`, ...

`hgrid` ⟨*list of relative vertical coordinates*⟩. Draws horizontal lines with respect to grid parameters.

`vgrid` ⟨*list of relative horizontal coordinates*⟩. Draws vertical lines with respect to grid parameters.

`plot(`*expr* ⟨*formula*⟩`,` $x_{\min}$`,` $x_{\max}$`,` n_{step}`)`. Control sequence. It draws the graph defined by the formula andalso stores it into `lastpath` MetaPost path variable (may be interesting for intersections, but beware of relative units).

`noplot(`*expr* ⟨*formula*⟩`,` $x_{\min}$`,` $x_{\max}$`,` n_{step}`)`. Control sequence. It just computes the graph defined by the formula and stores it into `lastpath` MetaPost path variable with relative units.

`abline(`*expr* a`,` b`)`. Control sequence. Draws the line $y = a + bx$ within the range of the current figure.

`normal(`*expr* μ`,` σ`,` $x_{\min}$`,` $x_{\max}$`,` n_{step}`)`. Control sequence. It plots the density function of $\mathcal{N}(\mu, \sigma)$, the normal distribution with mean μ and standard deviation σ,

$$x \longmapsto \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) \times \frac{1}{\sqrt{2\pi}\,\sigma}\,.$$

One can scale the plot or use some specific color just by adding usual options like `yscaled x.tw` or `withcolor blue`.

§ 6. Graphical procedures and their parameters

`barchart` *suffix* x. Control sequence.

`barthickness`. Numeric parameter. Thickness of bars in barchart diagrams. It must be given in relative coordinates. When plotting a barchart of a non-numeric variable, the relative distance between each value is 1, so one has to take care of this fact when setting `barthickness` in this case.

`cumulativebarchart` *suffix* x. Control sequence.

`bounds` *text* t. Control sequence. This control sequence defines the bounds that may be used later on.

`boundsflag`. Boolean parameter. This boolean is set to `true` when invocating the `bounds` control sequence. It is reset to `false` as the so defined bounds are used by some forecoming control sequence.

`currentbounds`. Control sequence. This control sequence contains the bounds defined with the `bounds` control sequence.

`lastbounds`. Control sequence. This control sequence contains the last bounds that has been used and may be helpful for further treatment.

`sturges` ⟨*primary*⟩ *x*`.`

$$\text{sturges}(x) = \text{ceiling}\big(1 + 3.322\log(x)\big).$$

`histogram` *suffix x*`.` Control sequence.

`cumulativehistogram` *suffix x*`.` Control sequence.

`boxplot(`*suffix x*`)(`*expr a*`,` *b*`).` Control sequence. Draws a vertical boxplot of variable x with horizontal bounds (width) defined by a and b. The relevant informations given by this diagram are: the "minimum", the first quartile $q_{1/4}$, the median $q_{1/2}$, the third quartile $q_{3/4}$ and the "maximum". Quartiles are computed with respect to the whole data contained in x, but the extrema may omit too large values: when the boolean variable `tuckeyflag` is set to `true`, then lowerbound $= \max(\min(x), q_{1/4} - 1.5(q_{3/4} - q_{1/4}))$ and upperbound $= \min(\max(x), q_{3/4} + 1.5(q_{3/4} - q_{1/4}))$, else lowerbound $= \max(\min(x), q_{0.01})$ and upperbound $= \min(\max(x), q_{0.99})$. Values outside of this range are always plotted and their numeric values are given when the boolean `gmarkflag2` is true. The average *x*`.mean` is plotted, and its numerical value is given when `gmarkflag1` is true. The following numerical quantities are defined

$$x\texttt{.mean}, \quad x\texttt{.q1}, \quad x\texttt{.q2}, \quad x\texttt{.q3}, \quad x\texttt{.lmoustache}, \quad x\texttt{.umoustache}$$

(only if they are not already defined, think of conditional plot). There is also the `boxplotspread` numerical parameter. When it is set to a strictly positive value, then horizontal lines are drawn at the boundaries of the diagram spreading over $[a, b]$ by an amount given by this parameter (it is ugly be quite usual, man makes the diagram clearer).

`tuckeyflag`. Boolean parameter. *See* `boxplot`.

`piechart(`*suffix x*`)(`*expr* ⟨*center*⟩`,` ⟨*x radius*⟩`,` ⟨*y radius*⟩`,` ⟨*angle*⟩`).` Control sequence.

`piefactor`. Numeric parameter. The radius of every slice is equal to the radius given as parameter of `piechart` times (1 − `piefactor`). Every slice is flushed to the limits of the circle with radius given as parameter.

`gmarkflag[], gmarkloc[], gmarkangle[].`

`frequencymode`. Internal numeric parameter. 0 for probability or proportion, 1 for frequency, 2 for percent.

§ 7. Graphical options and their parameters

`options` ⟨*undelimited text*⟩`.`

`fillcolor`. Color parameter.

`fillcolors` ⟨*list of colors*⟩`.` Control sequence. It defines a list of colors that may be used of multicolored diagrams. If the list of colors is non-void, the color parameter `fillcolor` is set to be equal to the first color of the list (`fillcolor[0]`), otherwise `fillcolor[0]` is set to be equal to `fillcolor`; these colors are stored into a color row `fillcolor[]`. Starting a multicolored diagram, `fillcolor[0]` should always be reset to the current value of `fillcolor`. For manual use of the so defined color list, just set

`fillcolor := fillcolor[`*i*`]`.

See also the `gradient` control sequence.

`hatchbreadth`. Numeric parameter.

`hatchstep`. Numeric parameter.

`alphafill`. Predefined graphical option. In order to compare, say, two histograms, it is nice to put them one on the other. If their stripes are not filled, it's ok, if they are hatched, it's also ok. But when they are filled with plain solid colors, some stripes may mask some others totally. That's why we introduce some transparency (and also because it is funny). It suffices to type

```
options alphafill;
```

then automatic filling will be done using the `alphafill` control sequence whose syntax is strictly `alphafill` ⟨*cycle_path*⟩, i.e., it admits no options and one has to change manually `fillcolor` to change the color of the filling.

`fgalpha`. Numeric parameter. The parameter `fgalpha` is a number lying between 0 and 1. The closer `fgalpha` is to 0, the more transparent fillings are; the closer `alpha` is to 1, the more opaque fillings are.

`bgalpha`. Numeric parameter. The parameter `bgalpha` is a number lying between 0 and 1. It's role is to take the background color into account or not when performing `alphafill`. The closer `bgalpha` is to 0, the more background color dominates; the closer `bgalpha` is to 1, the less it dominates. We believe that one should have

$$0 < \texttt{fgalpha} \leq \texttt{bgalpha} \leq 1,$$

and that it's better to have `bgalpha` = 1 since there is no real reason for taking the background color into account here.

`autoframe`. Predefined graphical option.

`noframe`. Predefined graphical option.

`frameflag`. Boolean parameter.

`gridoptions(`*text*`)`.

`autogrid`. Predefined graphical option.

`autovgrid`. Predefined graphical option.

`autohgrid`. Predefined graphical option.

`nogrid`. Predefined graphical option.

`hgridflag`. Boolean. Tells if an horizontal grid should be drawn once the figure achieved. It is set to true when using the option `autohgrid` or `autogrid`.

`vgridflag`. Boolean. Tells if an vertical grid should be drawn once the figure achieved. It is set to true when using the option `autovgrid` or `autogrid`.

`gridcolor`. Color. Color of the grids.

`gridbreadth`. Numeric. Thickness of the pen used to draw horizontal and vertical grids.

`hgridstep`. Numeric. Recommanded horizontal spacing between the vertical lines of the optional horizontal grid.

`vgridstep`. Numeric. Recommanded vertical spacing between the horizontal lines of the optional vertical grid.

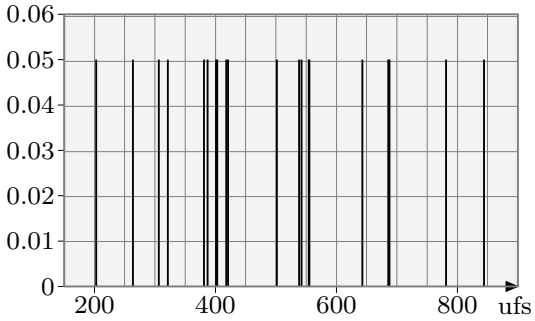
`gridstep`. Macro. `gridstep := 5mm` is equivalent to `hgridstep := vgridstep := 5mm`.

`opacity`. Predefined graphical option.

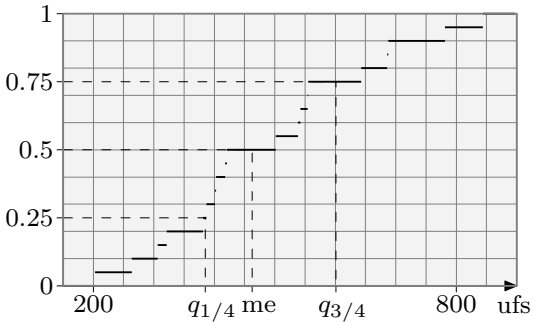
`noopacity`. Predefined graphical option.

`opacityflag`. Boolean parameter.

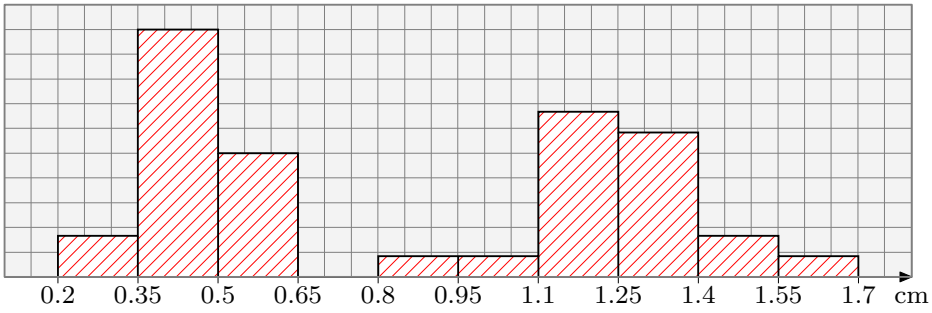
§ 8. Samples



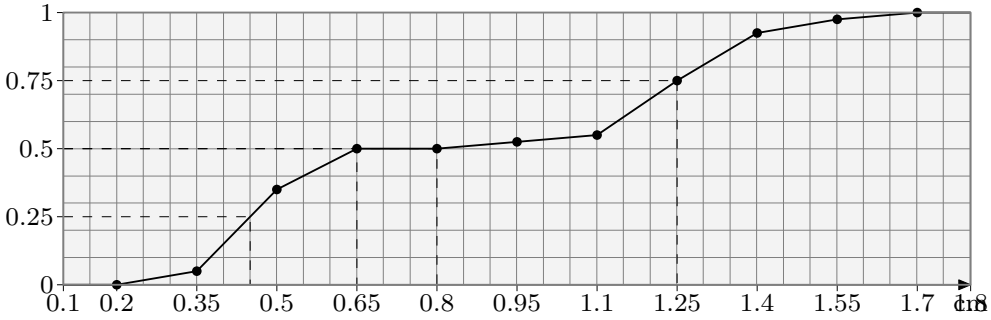
`test-mps.1`



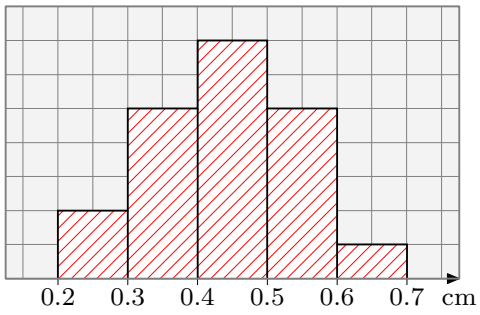
`test-mps.2`



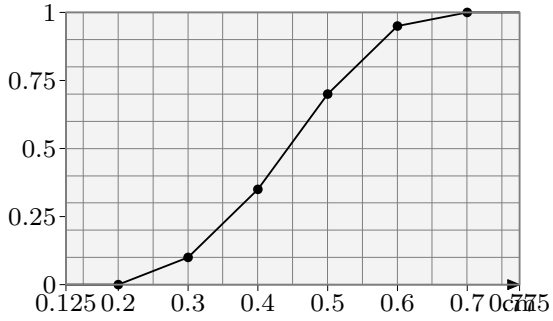
`test-mps.3`



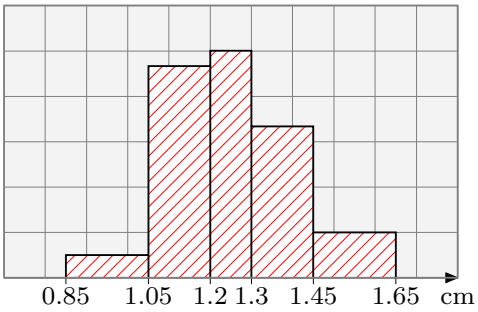
`test-mps.4`



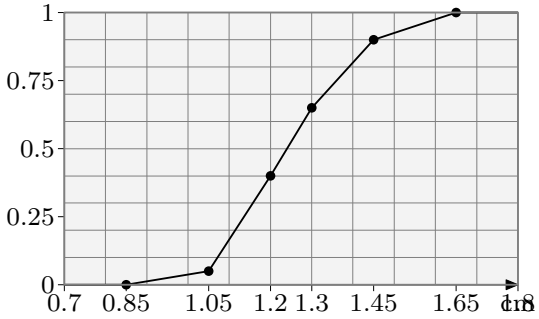
`test-mps.5`



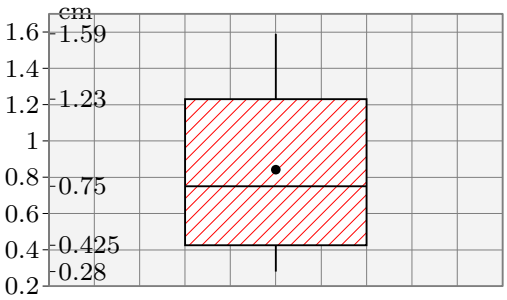
`test-mps.6`



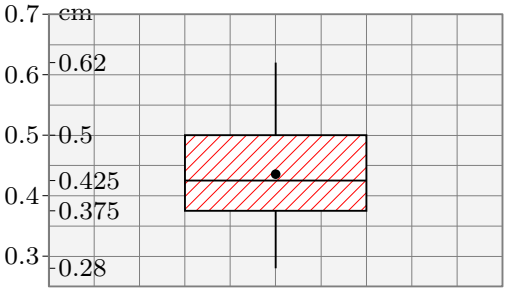
test-mps.7



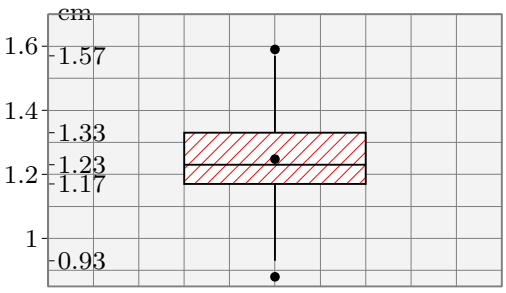
test-mps.8



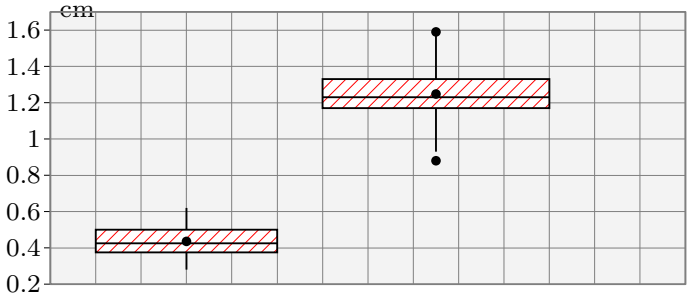
test-mps.9



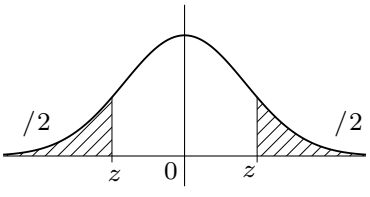
test-mps.10



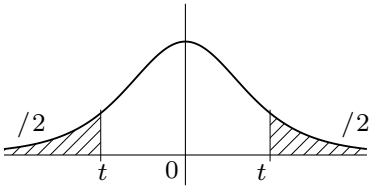
test-mps.11



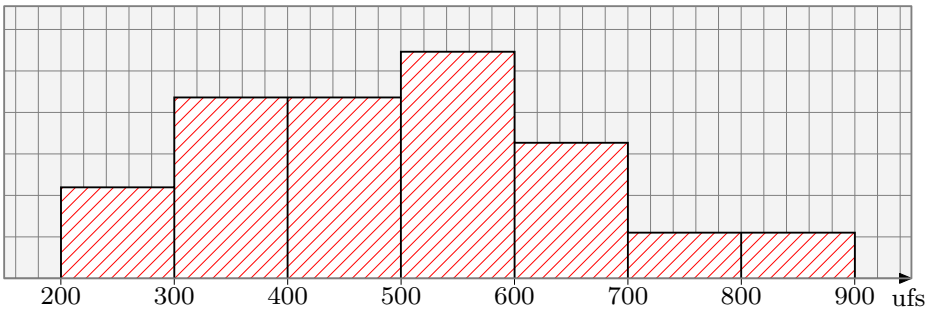
test-mps.12



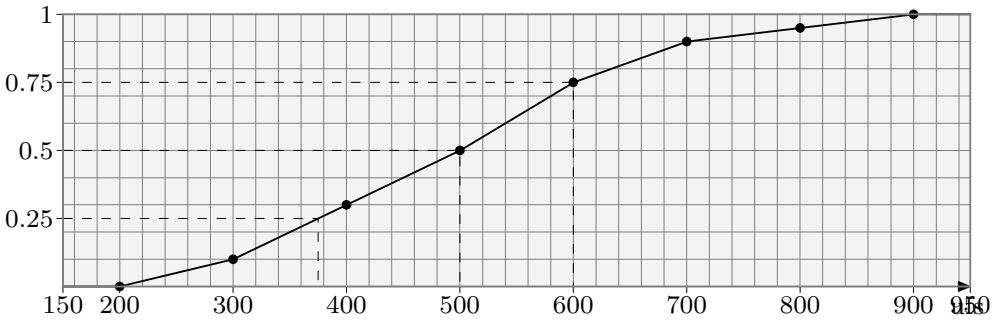
test-mps.13



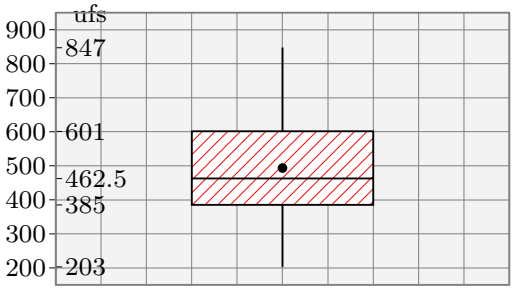
test-mps.14



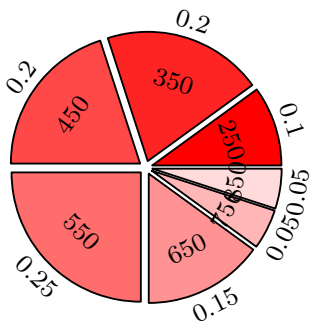
test-mps.15



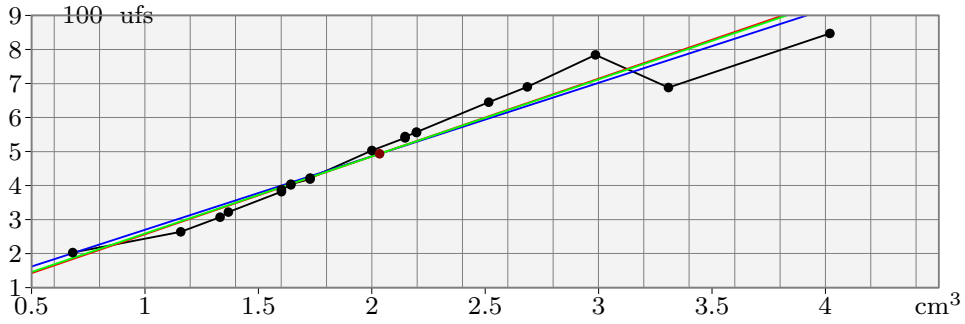
test-mps.16



test-mps.17

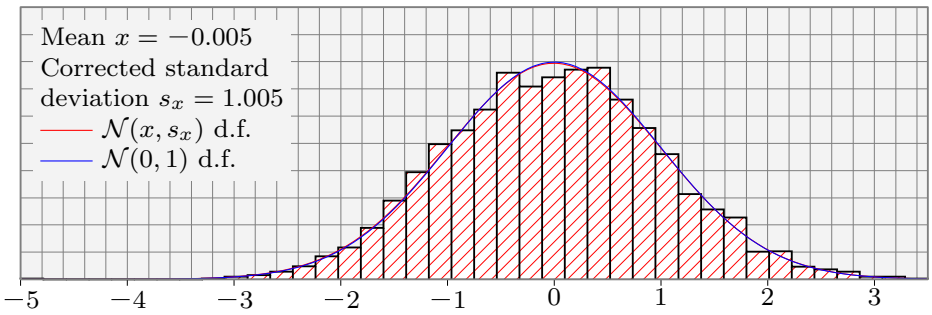


test-mps.18



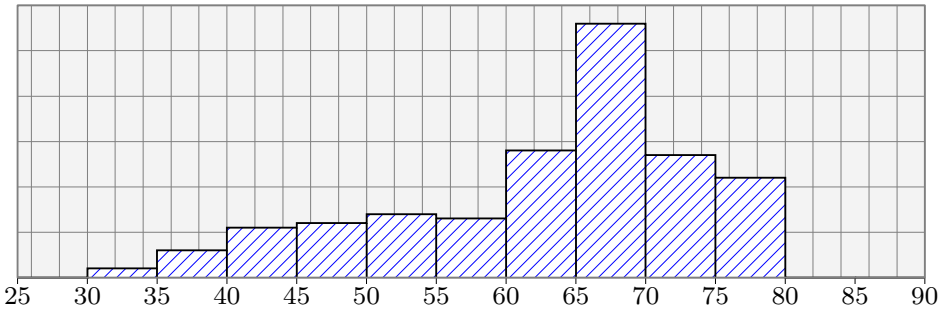
test-mps.19

4094 outcomes produced by `normaldeviate`



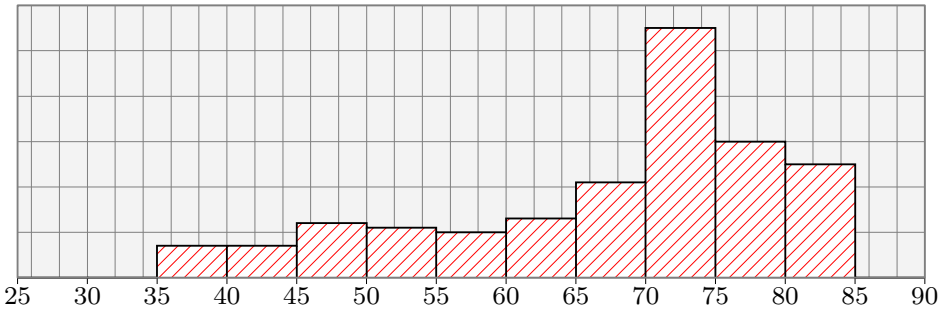
`test-mps.20`

Men life expectancy by country, OMS (WHO) 2001

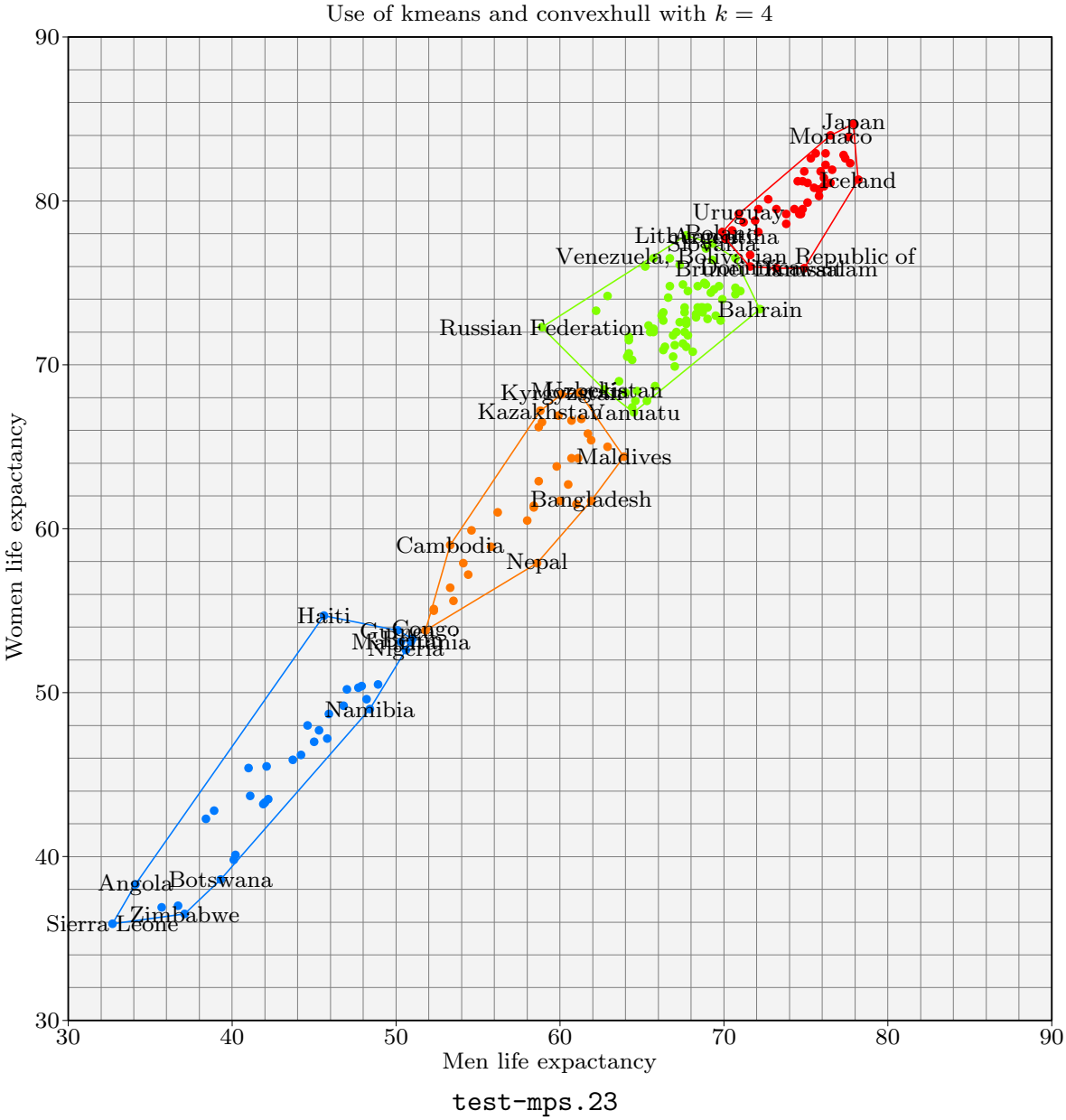


`test-mps.21`

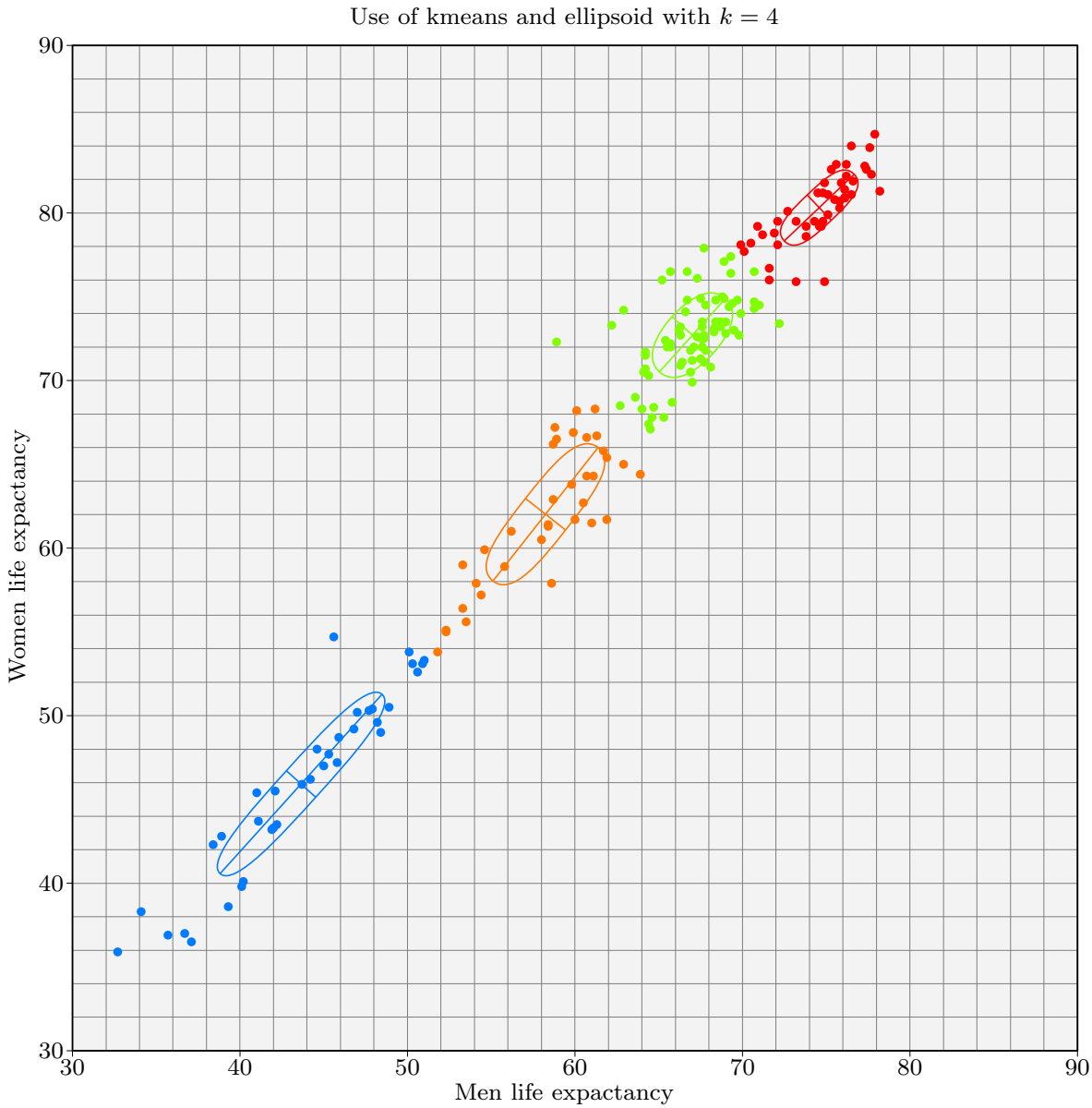
Women life expectancy by country, OMS (WHO) 2001



`test-mps.22`


Use of kmeans and convexhull with $k = 4$
Women life expactancy
Men life expactancy
30
40
50
60
70
80
90
Japan
Monaco
Iceland
Uruguay
Venezuela, Bolivarian Republic of
Bahrain
Russian Federation
Kazakhstan
Vanuatu
Maldives
Bangladesh
Cambodia
Nepal
Haiti
Namibia
Angola
Botswana
Zimbabwe
Sierra Leone

`test-mps.23`


Use of kmeans and ellipsoid with $k = 4$
Women life expactancy
Men life expactancy
90
80
70
60
50
40
30
30
40
50
60
70
80
90

`test-mps.24`

MPS-MATH.MP

MetaPost Promotion Pages

Anthony Phan

Introduction

The file `mps-math.mp` is a MetaPost library of rather common real valued mathematical functions of one real variable. It has been designed as part of the `mps` suite and, thus, it is quite oriented to probabilistic computations.

Since MetaPost numerical computations are rather unaccurate and limited—let us recall that it is based on a fixed point arithmetic with a precision at most equal to 1/65536 and that assignable values cannot excess $4096 - 1/65536$ in absolute value—, one cannot really expect to have a "scientific" calculator based on MetaPost.

Nevertheless, MetaPost capacities appear to be large enough for academic purpose: drawing graphs, making rather simple computations. Providing `mps-math.mp` as a standalone MetaPost input file may have two interests: saving time of MetaPost users and enabling some enhancements of `mps-math.mp`.

§ 1. Numerical computations

`solve(`*expr* ⟨*formula*⟩`,` ⟨*lowerbound*⟩`,` ⟨*upperbound*⟩`).` This control sequence searches for zeros of $x \mapsto f(x)$ with $x \in [\langle lowerbound\rangle, \langle upperbound\rangle]$ by the bisection or dichotomy method. The ⟨*formula*⟩ is a string defining an expression whose only variable or unknown value is `x` (for instance `zero = solve("(x**2)-1", -2, 2)`). It stops when accuracy is reached and then returns a number.

`solve.secant(`*expr* ⟨*formula*⟩`,` ⟨*lowerbound*⟩`,` ⟨*upperbound*⟩`).` This is the same control sequence as before except that the suffix `secant` forces to use the secant method for finding a zero. It stops when accuracy is reached and then returns a number.

`integrate(`*expr* ⟨*formula*⟩`,` ⟨*lowerbound*⟩`,` ⟨*upperbound*⟩`).` The meaning of this control sequence is obvious, its syntax is the same as before. It stops when accuracy is reached and then returns a number.

§ 2. Usual mathematical functions

The most usual mathematical functions are implemented into `mps-math.mp` and described thereafter. The syntax is the most common one. The descriptions show domains and ranges, usual/standard notations, and also the meaning of each function. Arguments too close to the bounds of the domains may lead to errors related to MetaPost capabilities; arguments outside of the domains may give error or unproper results.

`PI (or pi).` Internal numeric whose value is $3.141592654 \approx \pi$.

`STPI.` Internal numeric whose value is $2.506628275 \approx \sqrt{2\pi}$.

`exp` ⟨*primary*⟩ x. Usual exponential function

$$\mathbf{R} \longrightarrow \mathbf{R}_+^*, \qquad x \longmapsto \exp(x) = \mathrm{e}^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}\,.$$

`ln` ⟨*primary*⟩ x. Usual (natural or Neperian) logarithm function

$$\mathbf{R}_+^* \longrightarrow \mathbf{R}, \qquad x \longmapsto \ln(x) = \mathrm{Log}(x) = \exp^{-1}(x).$$

`log` ⟨*primary*⟩ x. Decimal logarithm function

$$\mathbf{R}_+^* \longrightarrow \mathbf{R}, \qquad x \longmapsto \log(x) = \mathrm{Log}_{10}(x).$$

`cosh` ⟨*primary*⟩ x. Hyperbolic cosine function

$$\mathbf{R} \longrightarrow [1, +\infty[, \qquad x \longmapsto \cosh(x) = \mathrm{ch}(x) = \frac{\mathrm{e}^x + \mathrm{e}^{-x}}{2} = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!}\,.$$

`sinh` ⟨*primary*⟩ x. Hyperbolic sine function

$$\mathbf{R} \longrightarrow \mathbf{R}, \qquad x \longmapsto \sinh(x) = \mathrm{sh}(x) = \frac{\mathrm{e}^x - \mathrm{e}^{-x}}{2} = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!}\,.$$

`tanh` ⟨*primary*⟩ x. Hyperbolic tangent function

$$\mathbf{R} \longrightarrow]-1, 1[, \qquad x \longmapsto \tanh(x) = \mathrm{th}(x) = \frac{\mathrm{e}^x - \mathrm{e}^{-x}}{\mathrm{e}^x + \mathrm{e}^{-x}}\,.$$

`coth` ⟨*primary*⟩ x. Hyperbolic cotangent function

$$\mathbf{R} \longrightarrow]-\infty, -1[\cup]1, +\infty[, \qquad x \longmapsto \coth(x) = \frac{\mathrm{e}^x + \mathrm{e}^{-x}}{\mathrm{e}^x - \mathrm{e}^{-x}}\,.$$

`arccosh` ⟨*primary*⟩ x. Inverse hyperbolic cosine function

$$[1, +\infty[\longrightarrow \mathbf{R}_+, \qquad x \longmapsto \operatorname{arc\,cosh}(x) = \arg \mathrm{ch}(x) = \ln\Big(x + \sqrt{x^2 - 1}\Big).$$

`arcsinh` ⟨*primary*⟩ x. Inverse hyperbolic sine function

$$\mathbf{R} \longrightarrow \mathbf{R}, \qquad x \longmapsto \operatorname{arc\,sinh}(x) = \arg \mathrm{sh}(x) = \ln\Big(x + \sqrt{x^2 + 1}\Big).$$

`arctanh` ⟨*primary*⟩ x. Inverse hyperbolic tangent function

$$]-1, 1[\longrightarrow \mathbf{R}, \qquad x \longmapsto \operatorname{arc\,tanh}(x) = \arg \mathrm{th}(x) = \frac{1}{2}\left(\ln\Big(\frac{1+x}{1-x}\Big)\right).$$

`arccoth` ⟨*primary*⟩ x. Inverse hyperbolic cotangent function

$$]-\infty, -1[\cup]1, +\infty[\longrightarrow \mathbf{R}^*, \quad x \longmapsto \operatorname{arc\,coth}(x) = \arg \coth(x) = \operatorname{arc\,tanh}(1/x).$$

`cos` ⟨*primary*⟩ x. Circular cosine function

$$\mathbf{R} \longrightarrow [-1, 1], \qquad x \longmapsto \cos(x) = \frac{\mathrm{e}^{\mathrm{i}x} + \mathrm{e}^{-\mathrm{i}x}}{2} = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}\,.$$

`sin` ⟨*primary*⟩ x. Circular sine function

$$\mathbf{R} \longrightarrow [-1, 1], \qquad x \longmapsto \sin(x) = \frac{\mathrm{e}^{\mathrm{i}x} - \mathrm{e}^{-\mathrm{i}x}}{2\mathrm{i}} = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}\,.$$

`tan` ⟨*primary*⟩ *x*. Circular tangent function

$$\mathbf{R} \setminus (\pi/2 + \pi\mathbf{Z}) \longrightarrow \mathbf{R}, \qquad x \longmapsto \tan(x) = \mathrm{tg}(x) = \frac{\sin(x)}{\cos(x)}\,.$$

`cot` ⟨*primary*⟩ *x*. Circular cotangent function

$$\mathbf{R} \setminus \pi\mathbf{Z} \longrightarrow \mathbf{R}, \qquad x \longmapsto \cot(x) = \mathrm{cotg}(x) = \frac{\cos(x)}{\sin(x)}\,.$$

`arccos` ⟨*primary*⟩ *x*. Inverse circular cosine function

$$[-1,1] \longrightarrow [0,\pi], \qquad x \longmapsto \arccos(x) = \cos^{-1}(x).$$

`arcsin` ⟨*primary*⟩ *x*. Inverse circular sine function

$$[-1,1] \longrightarrow [-\pi/2,\pi/2], \qquad x \longmapsto \arcsin(x) = \sin^{-1}(x).$$

`arctan` ⟨*primary*⟩ *x*. Inverse circular tangent function

$$\mathbf{R} \longrightarrow \,]-\pi/2,\pi/2[, \qquad x \longmapsto \arctan(x) = \mathrm{arc\,tg}(x) = \arcsin\left(\frac{x}{\sqrt{x^2+1}}\right).$$

`arccot` ⟨*primary*⟩ *x*. Inverse circular cotangent function

$$\mathbf{R} \longrightarrow \,]0,\pi[, \qquad x \longmapsto \mathrm{arc\,cot}(x) = \mathrm{arc\,cotg}(x) = \arccos\left(\frac{x}{\sqrt{x^2+1}}\right).$$

Other usual functions are already implemented into (plain-)MetaPost: `abs`, `floor`, `ceiling`, `round`, `sqrt`, `takepower of`, ...

§ 3. Continuous Probability distributions

`lnGamma(`*expr x*`)`. Logarithm of the Eulerian Gamma function, $\ln(\Gamma(x))$.

`Gamma(`*expr x*`)`. Eulerian Gamma function

$$\mathbf{R}_+^* \longrightarrow [\sqrt{\pi},+\infty[, \qquad x \longmapsto \Gamma(x) = \int_0^{+\infty} t^{x-1}\,\mathrm{e}^{-t}\,\mathrm{d}t.$$

Remember that for $n \in \mathbf{N}$, $n! = \Gamma(n+1)$.

`Beta(`*expr x*`,` *y*`)`. Eulerian Beta function

$$(\mathbf{R}_+^*)^2 \longrightarrow \mathbf{R}_+^*, \qquad x \longmapsto \mathrm{B}(x,y) = \int_0^1 u^{x-1}(1-u)^{y-1}\,\mathrm{d}u = \frac{\Gamma(x)\times\Gamma(y)}{\Gamma(x+y)}\,.$$

`gammapdf(`*expr x*`,` *a*`,` *λ*`)`. Probability density function of the Gamma distribution with parameters $a > 0$, $\lambda > 0$.

$$\mathbf{R} \longrightarrow \mathbf{R}_+, \qquad x \longmapsto \mathbf{1}_{\mathbf{R}_+}(x)\frac{\lambda^a}{\Gamma(a)}\,x^{a-1}\,\mathrm{e}^{-\lambda x}.$$

`gammacdf(`*expr x*`,` *a*`,` *λ*`)`. Cumulative distribution function of the Gamma distribution with parameter $a > 0$, $\lambda > 0$.

$$\mathbf{R} \longrightarrow [0,1[, \qquad x \longmapsto \mathbf{1}_{\mathbf{R}_+}(x)\frac{\lambda^a}{\Gamma(a)}\int_0^x t^{a-1}\,\mathrm{e}^{-\lambda t}\,\mathrm{d}t.$$

`gammaicdf(`*expr* p`, `a`, `λ`)`. Inverse cumulative distribution function of the Gamma distribution with parameter $a > 0$, $\lambda > 0$.

$$[0,1[\longrightarrow \mathbf{R}_+, \qquad p \longmapsto \text{gammaicdf}(p, a, \lambda).$$

It is set to 0 for $p <$ `accuracy` and to `infinity` for $p > 1 -$ `accuracy`.

`normlimit`. Numerical parameter for normal computations: if X is a random variable with law $\mathcal{N}(0,1)$, the normal distribution with mean 0 and standard deviation 1, then $\mathbf{P}\{X \geq$ `normlimit`$\} = \mathbf{P}\{X \leq -$`normlimit`$\} \approx 0$. Its value is (reasonably) set to `normlimit` $= 4$.

`normalpdf(`*expr* x`)`. Probability density function of $\mathcal{N}(0,1)$.

$$\mathbf{R} \longrightarrow \mathbf{R}_+, \qquad x \longmapsto \mathrm{e}^{-x^2/2}/\sqrt{2\pi}.$$

`normalcdf` ⟨*primary*⟩ x`.` Cumulative distribution function of $\mathcal{N}(0,1)$ computed with the so-called Hastings best approximation method. To be experimented and improved. It is is supposed to replace the following implementation which is rather heavy. One can set `def normalcdf = normalcdf_ enddef;` for instance for computation of cdf or icdf with respect to a method or another.

`normalcdf_` ⟨*primary*⟩ x`.` Cumulative distribution function of $\mathcal{N}(0,1)$.

$$\mathbf{R} \longrightarrow]0,1[, \qquad x \longmapsto \Phi(x) = \int_{-\infty}^{x} \mathrm{e}^{-z^2/2}\,\frac{\mathrm{d}z}{\sqrt{2\pi}} = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n\, x^{2n+1}}{(2n+1) \times n!}\,.$$

It is computed with its associated power series when $|x| <$ `normlimit`, and set to 0 or 1 otherwise.

`normalcdf__` ⟨*primary*⟩ x`.` Cumulative distribution function of $\mathcal{N}(0,1)$. It is just another implementation of the previous function with a Gamma cumulative distribution function since

$$\Phi(x) = \frac{1}{2}\left(\text{sgn}(x) \times \text{gammacdf}(x^2/2, 1/2) + 1, 1\right), \qquad \text{for all } x \in \mathbf{R}.$$

`normalicdf` ⟨*primary*⟩ p`.` Inverse cumulative distribution function of $\mathcal{N}(0,1)$.

$$]0,1[\longrightarrow \mathbf{R}, \qquad p \longmapsto \text{normalicdf}(p) = \Phi^{-1}(p).$$

It is set to $\pm$`infinity` for p outside of]`accuracy`, $1 -$ `accuracy`[.

Remark. — Normal distributions with mean m and standard deviation σ are not inplemented sine they an be easily derived from the standard normal distribution. For instance, one can set

```
vardef gaussianpdf(expr x, m, sigma) = normalpdf((x-m)/sigma)/sigma enddef;
vardef gaussiancdf(expr x, m, sigma) = normalcdf((x-m)/sigma) enddef;
vardef gaussianicdf(expr p, m, sigma) = sigma*normalicdf(p)+m enddef;
```

in order to get the probability, cumulative, inverse cumulative distribution functions of the $\mathcal{N}(m, \sigma^2)$ distribution with $m \in \mathbf{R}$ and $\sigma > 0$.

`chisquarepdf(`*expr* x`, `ν`)`. Probability density function of $\chi^2(\nu)$, the chi-square (Pearson) distribution with $\nu > 0$ degrees of freedom.

$$\mathbf{R} \longrightarrow \mathbf{R}_+, \qquad x \longmapsto \mathbf{1}_{\mathbf{R}_+}(x)\,\frac{x^{\nu/2-1}\mathrm{e}^{-x/2}}{2^{\nu/2}\,\Gamma(\nu/2)}\,.$$

`chisquarecdf(`*expr* x`,` ν`)`. Cumulative distribution function of $\chi^2(\nu)$.

$$\mathbf{R} \longrightarrow \mathbf{R}_+, \qquad x \longmapsto \mathbf{1}_{\mathbf{R}_+}(x) \int_0^x t^{\nu/2-1} \mathrm{e}^{-t/2} \, \frac{\mathrm{d}t}{2^{\nu/2}\,\Gamma(\nu/2)} \, .$$

`chisquareicdf(`*expr* p`,` ν`)`. Inverse cumulative distribution function of $\chi^2(\nu)$.

$$[0,1[\longrightarrow \mathbf{R}_+, \qquad p \longmapsto \mathrm{chisquareicdf}(p,\nu).$$

It is set to 0 for $p <$ `accuracy` and to `infinity` for $p > 1 -$ `accuracy`.

`betapdf(`*expr* x`,` a`,` b`)`. Probability density function of the Beta distribution with parameters $a > 0$ and $b > 0$.

$$\mathbf{R} \longrightarrow \mathbf{R}_+, \qquad x \longmapsto \mathbf{1}_{[0,1]}(x)\, x^{a-1}(1-x)^{b-1}/\mathrm{B}(a,b).$$

`betacdf(`*expr* x`,` a`,` b`)`. Cumulative distribution function of the Beta distribution with parameters $a > 0$ and $b > 0$.

$$\mathbf{R} \longrightarrow \mathbf{R}_+, \qquad x \longmapsto \int_0^x \mathbf{1}_{[0,1]}(u)\, u^{a-1}(1-u)^{b-1} \, \frac{\mathrm{d}u}{\mathrm{B}(a,b)} \, .$$

`betaicdf(`*expr* p`,` a`,` b`)`. Inverse cumulative distribution function of the Beta distribution with parameters $a > 0$ and $b > 0$.

$$[0,1] \longrightarrow \mathbf{R}_+, \qquad p \longmapsto \mathrm{betaicdf}(p,a,b).$$

It is set to 0 for $p <$ `accuracy` and to 1 for $p > 1 -$ `accuracy`.

`studentpdf(`*expr* x`,` ν`)`. Probability density function of $\mathcal{T}(\nu)$, the Student distribution with $\nu > 0$ degrees of freedom.

$$\mathbf{R} \longrightarrow \mathbf{R}_+, \qquad x \longmapsto \frac{1}{\sqrt{\nu}\,\mathrm{B}(\nu/2,1/2)} \left(1 + \frac{x^2}{\nu}\right)^{-(\nu+1)/2} \, .$$

`studentcdf(`*expr* x`,` ν`)`. Cumulative distribution function of $\mathcal{T}(\nu)$.

$$\mathbf{R} \longrightarrow \mathbf{R}_+, \qquad x \longmapsto \int_{-\infty}^x \left(1 + \frac{z^2}{\nu}\right)^{-(\nu+1)/2} \frac{\mathrm{d}z}{\sqrt{\nu}\,\mathrm{B}(\nu/2,1/2)}.$$

`studenticdf(`*expr* p`,` ν`)`. Inverse cumulative distribution function of $\mathcal{T}(\nu)$.

$$]0,1[\longrightarrow \mathbf{R}, \qquad p \longmapsto \mathrm{studenticdf}(p,\nu).$$

It is set to $\pm$`infinity` for p outside of]`accuracy`, $1 -$ `accuracy`[.

`fisherpdf(`*expr* x`,` ν_1`,` ν_2`)`. Probability density function of $\mathcal{F}(\nu_1,\nu_2)$, the Fisher distribution with $\nu_1 > 0$ and $\nu_2 > 0$ degrees of freedom (ν_1 is the numerator degree of freedom, ν_2 the demominator degree of freedom).

$$\mathbf{R} \longrightarrow \mathbf{R}_+, \qquad x \longmapsto \mathbf{1}_{\mathbf{R}_+}(x) \, \frac{(\nu_1/\nu_2)^{\nu_1/2}\, x^{\nu_1/2-1}}{\mathrm{B}(\nu_1/2,\nu_2/2)(1 + x \times \nu_1/\nu_2)^{(\nu_1+\nu_2)/2}}.$$

`fishercdf(`*expr* x`,` ν_1`,` ν_2`)`. Cumulative distribution function of $\mathcal{F}(\nu_1,\nu_2)$.

$$\mathbf{R} \longrightarrow \mathbf{R}_+, \qquad x \longmapsto \mathbf{1}_{\mathbf{R}_+}(x) \int_0^x \frac{(\nu_1/\nu_2)^{\nu_1/2}\, z^{\nu_1/2-1}}{\mathrm{B}(\nu_1/2,\nu_2/2)(1 + z \times \nu_1/\nu_2)^{(\nu_1+\nu_2)/2}} \, \mathrm{d}z.$$

`fishericdf(`*expr* p`,` ν_1`,` ν_2`)`. Inverse cumulative distribution function of $\mathcal{F}(\nu_1,\nu_2)$.

$$[0,1[\longrightarrow \mathbf{R}_+, \qquad p \longmapsto \mathrm{fishericdf}(p,\nu_1,\nu_2).$$

It is set to 0 for $p <$ `accuracy` and to `infinity` for $p > 1 -$ `accuracy`.

§ 4. Discrete Probability distributions

About quantiles, please note that $q_p = k + 0.5$ when $F(k) = p$ for $k \in \mathbf{N}$.

`poissonpdf(`*expr* x`, `λ`)`. Probability distribution function of $\mathcal{P}(\lambda)$, the Poisson distribution with parameter $\lambda \geq 0$.

$$\mathbf{R} \longrightarrow [0,1], \qquad x \longmapsto \begin{cases} \mathrm{e}^{-\lambda}\,\lambda^x/x! & \text{if } x \in \mathbf{N}, \\ 0 & \text{otherwise.} \end{cases}$$

`poissoncdf(`*expr* x`, `λ`)`. Cumulative distribution function of $\mathcal{P}(\lambda)$.

`poissonicdf(`*expr* p`, `λ`)`. Inverse cumulative distribution function of $\mathcal{P}(\lambda)$.

`binomialpdf(`*expr* x`, `n`, `π`)`. Probability distribution function of $\mathcal{B}(n,\pi)$, the binomial distribution with parameters $n \in \mathbf{N}^*$ and $\pi \in [0,1]$.

$$\mathbf{R} \longrightarrow [0,1], \qquad x \longmapsto \begin{cases} C_n^x\,\pi^x(1-\pi)^{n-x} & \text{if } x \in \{0,1,\ldots,n\}, \\ 0 & \text{otherwise.} \end{cases}$$

`binomialcdf(`*expr* x`, `n`, `π`)`. Cumulative distribution function of $\mathcal{B}(n,\pi)$.

`binomialicdf(`*expr* p`, `n`, `π`)`. Inverse cumulative distribution function of $\mathcal{B}(n,\pi)$.

`geometricpdf(`*expr* x`, `π`)`. Probability distribution function of $\mathcal{G}(\pi)$, the geometrical distribution with parameter $\pi \in [0,1]$. Describe the law of the first success rank in an infinitely repeated Bernoulli trial with parameter $\pi \in [0,1]$. Thus, it is given by

$$\mathbf{R} \longrightarrow [0,1], \qquad x \longmapsto \begin{cases} \pi(1-\pi)^{x-1} & \text{if } x \in \{1,2,3,\ldots\}, \\ 0 & \text{otherwise.} \end{cases}$$

`geometriccdf(`*expr* x`, `π`)`. Cumulative distribution function of $\mathcal{G}(\pi)$. It returns the sum up to x of the previous probabilities. Thus it is given by

$$\mathbf{R} \longrightarrow [0,1], \qquad x \longmapsto \begin{cases} 1-(1-\pi)^{\operatorname{floor} x} & \text{if } x \geq 1, \\ 0 & \text{otherwise.} \end{cases}$$

`geometricicdf(`*expr* p`, `π`)`. Inverse cumulative distribution function of $\mathcal{G}(\pi)$.

$$[0,1] \longrightarrow \{1, 1.5, 2, 2.5, 3, 3.5, \ldots\}, \qquad p \longmapsto \text{geometricicdf}(p,\pi).$$

`negativebinomialpdf(`*expr* x`, `n`, `π`)`. Probability distribution function of the negative binomial ditribution with parameters $n \in \mathbf{N}^*$ and $\pi \in [0,1]$. Describe the law of the n-th success rank, $n \in \mathbf{N}^*$, in an infinitely repeated Bernoulli trial with parameter $\pi \in [0,1]$. Thus, it is given by

$$\mathbf{R} \longrightarrow [0,1], \qquad x \longmapsto \begin{cases} C_{x-1}^{n-1}\,\pi^n(1-\pi)^{x-n} & \text{if } x \in \{n, n+1, n+2, \ldots\}, \\ 0 & \text{otherwise.} \end{cases}$$

One can remark that the negative binomial distributions for $n = 1$ are just the corresponding geometric ones. It is certainly better when $n = 1$ to use functions related to geometric distributions instead of negative binomial distributions related ones.

`negativebinomialcdf(`*expr* x`, `n`, `π`)`. Cumulative distribution function of the negative binomial ditribution with parameters $n \in \mathbf{N}^*$ and $\pi \in [0,1]$. One can easily prove that it is given by

$$\mathbf{R} \longrightarrow [0,1], \qquad x \longmapsto 1 - \text{binomialcdf}(n-1, \operatorname{floor} x, \pi).$$

`negativebinomialicdf(`*expr* p`,` n`,` π`).` Inverse cumulative distribution function of the negative binomial ditribution with parameters $n \in \mathbf{N}^*$ and $\pi \in [0,1]$.

$$[0,1] \longrightarrow \{n, n+0.5, n+1, n+1.5, \ldots\}, \qquad p \longmapsto \text{negativebinomialicdf}(p,n,\pi).$$

`hypergeometricpdf(`*expr* x`,` N`,` n`,` π`).` Probability distribution function of $\mathcal{H}(N,n,\pi)$, the hypergeometrical distribution with parameters $N \geq n \in \mathbf{N}^*$ and $\pi \in [0,1]$. One should have $N\pi \in \mathbf{N}$.

$$\mathbf{R} \longrightarrow [0,1],$$
$$x \longmapsto \begin{cases} \dfrac{C_{N\pi}^{x}\, C_{N(1-\pi)}^{n-x}}{C_N^n} & \text{if } x \in \{\max(0, n-N(1-\pi)), \ldots, \min(n, N\pi)\}, \\ 0 & \text{otherwise.} \end{cases}$$

`hypergeometriccdf(`*expr* x`,` N`,` n`,` π`).` Cumulative distribution function of $\mathcal{H}(N,n,\pi)$.

`hypergeometricicdf(`*expr* p`,` N`,` n`,` π`).` Inverse cumulative distribution function of $\mathcal{H}(N,n,\pi)$.

§ 5. Rather specific Probability distributions

`kolmogorovcdf(`*expr* x`,` n`).` Cumulative distribution function of Kolmogorov distributions. These are the famous probability distributions involved in Kolmogorov–Smirnov (two-sided) Goodness of Fit tests with statistic

$$K_n = \|F - F_n\|_\infty = \sup_{x\in\mathbf{R}} \big|F(x) - F_n(x)\big| = \max_{i=1}^{n} \big(F(X_{(i)}) - (i-1)/n\big) \vee \big(i/n - F(X_{(i)})\big).$$

Their computation is based on "Evaluating Kolmogorov's Distribution" by George Marsaglia and Wai Wan Tsang. Accuracy is of course not as good in MetaPost as it can be in C.

Remark. — Computations are done with some home made float format. Floats are represented as MetaPost pairs (x,y), x is the mantissa, y is the exponent, the basis is `fbasis` $= 64$. If $x \neq 0$, then $1 \leq |x| <$ `fbasis` where `fbasis` $= 64$, thus, in binary terms, x is coded with 6 digits point 16 digits. The exponent is computed in `epsilon` units, so its range is $\{-2^{28}-1, \ldots, 2^{28}-1\}$. This roughly allows to code non zero numbers with absolute value between $64^{-2^{28}-1} = 2^{-3\times(2^{30}-4)}$ and $64^{2^{28}} = 2^{3\times 2^{30}}$. The choice of `fbasis` $= 64$ is of course because $64 \times 64 = 4096$ the fixed upper limit of MetaPost fixed point computations (think of products). This is also fine because the lower limit is 2^{-16}. Numerical overflows should not happen with some normal use. What can happen are memory overflows due to Marsaglia & Al. method.

`kolmogorovicdf(`*expr* p`,` n`).` Inverse cumulative distribution function of Kolmogorov distributions. (Please do not use it since it is based on the bisection method or dichotomy.)

`klmcdf(`*expr* x`,` n`).` Cumulative distribution function of the limiting distributions associated to Kolmogorov distribution by Dudley's asymptotic formula (1964):

$$\lim_{n\to\infty} \mathbf{P}\big\{K_n \leq u/\sqrt{n}\big\} = 1 + 2\sum_{k=1}^{\infty} (-1)^k \exp\big(-2k^2u^2\big),$$

with some numerical adjustments (Stephens M.A., 1970), and other ones for small values and large ones.

`klmicdf(`*expr* p`,` n`)`. Inverse cumulative distribution function of the previous distributions.

`kpmcdf(`*expr* x`,` n`)`. Cumulative distribution function of the distribution involved in the one sided Goodness of Fit tests with statistic

$$K_n^+ = \sup_{x\in\mathbf{R}}\big(F(x) - F_n(x)\big) = \max_{1\le i\le n}\Big(\frac{i}{n} - F\big(X_{(i)}\big)\Big)$$

or

$$K_n^- = \sup_{x\in\mathbf{R}}\big(F_n(x) - F(x)\big) = \max_{1\le i\le n}\Big(F\big(X_{(i)}\big) - \frac{i-1}{n}\Big)$$

which share the same distribution: for $x \in [0,1]$,

$$\begin{aligned}\texttt{kpmcdf}(x,n) = \mathbf{P}\big\{K_n^{\pm} \le x\big\} &= x\sum_{0\le k\le nx}\binom{n}{k}(k/n - x)^k(x+1-k/n)^{n-k-1}\\ &= 1 - x\sum_{nx<k\le n}\binom{n}{k}(k/n - x)^k(x+1-k/n)^{n-k-1},\end{aligned}$$

the second formula being the one used for computations.

`kpmicdf(`*expr* p`,` n`)`. Inverse cumulative distribution function of the previous distributions.

`exponentialcdf(`*expr* x`,` λ`)`. And also `pdf` and `icdf`.

`cauchycdf(`*expr* x`,` a`)`. And also `pdf` and `icdf`.

`paretocdf(`*expr* x`,` $x_{\min}$`,` k`)`. And also `pdf` and `icdf`.

`GPacdf(`*expr* x`,` μ`,` σ`,` ξ`)`. And also `pdf` and `icdf`.

`gumbelcdf(`*expr* x`,` μ`,` σ`,` ξ`)`. And also `pdf` and `icdf`.

`frechetcdf(`*expr* x`,` μ`,` σ`,` α`)`. And also `pdf` and `icdf`.

`weibulcdf(`*expr* x`,` μ`,` σ`,` α`)`. And also `pdf` and `icdf`.

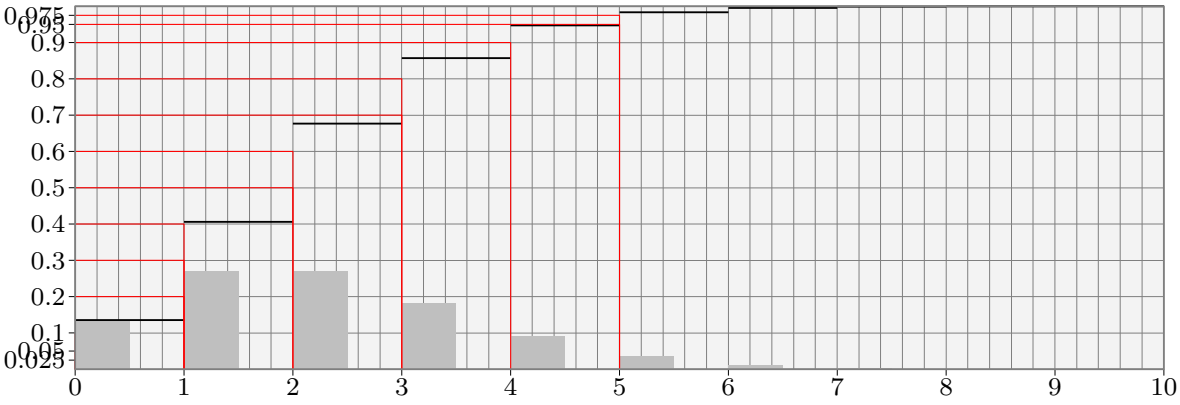
`GEVcdf(`*expr* x`,` μ`,` σ`,` ξ`)`. And also `pdf` and `icdf`.

`choose(`*expr* n`,` k`)`. It simply gives $C_n^k = \binom{n}{k} = n!/(k!\,(n-k)!)$ the usual binomial coefficients computed by rounding the righthand expression in the following identity:

$$C_n^k = \binom{n}{k} = \frac{n!}{k!\,(n-k)!} = \frac{n}{1}\times\frac{n-1}{2}\times\cdots\times\frac{n-k+1}{k}\,.$$

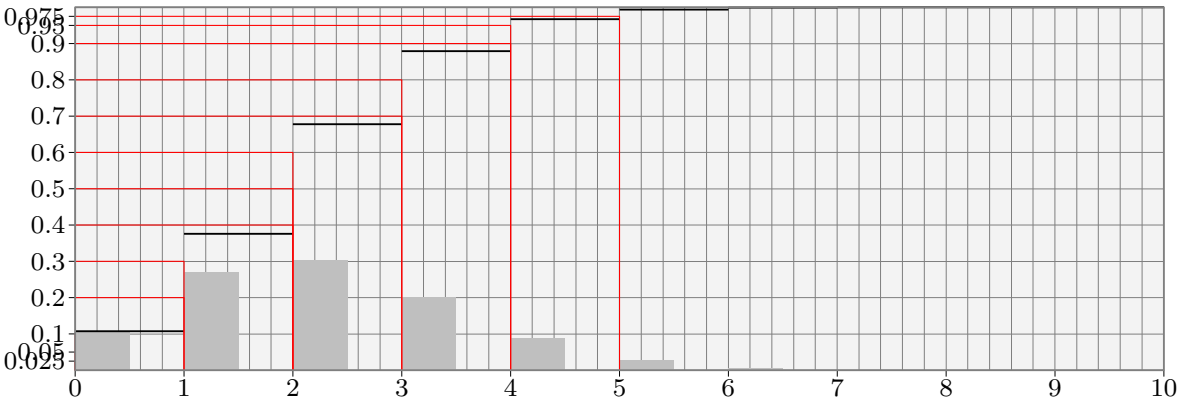
§ 6. Graphical tests

poisson distribution with parameter 2



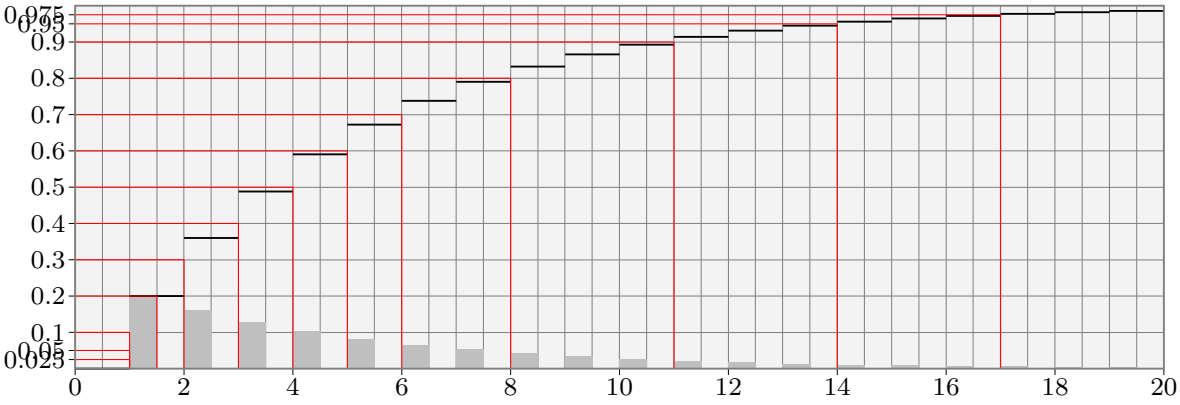
test-math.1

binomial distribution with parameters (10,0.2)



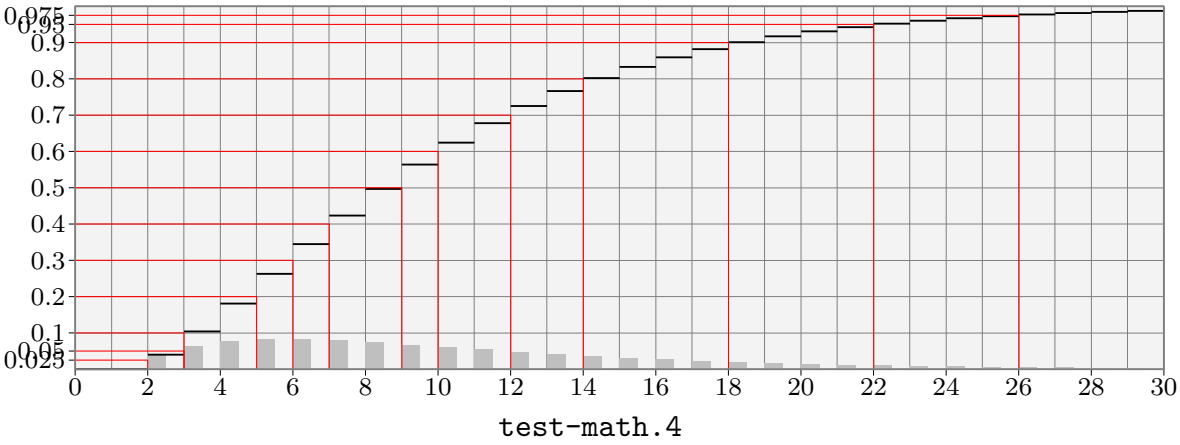
test-math.2

geometric distribution with parameter 0.2



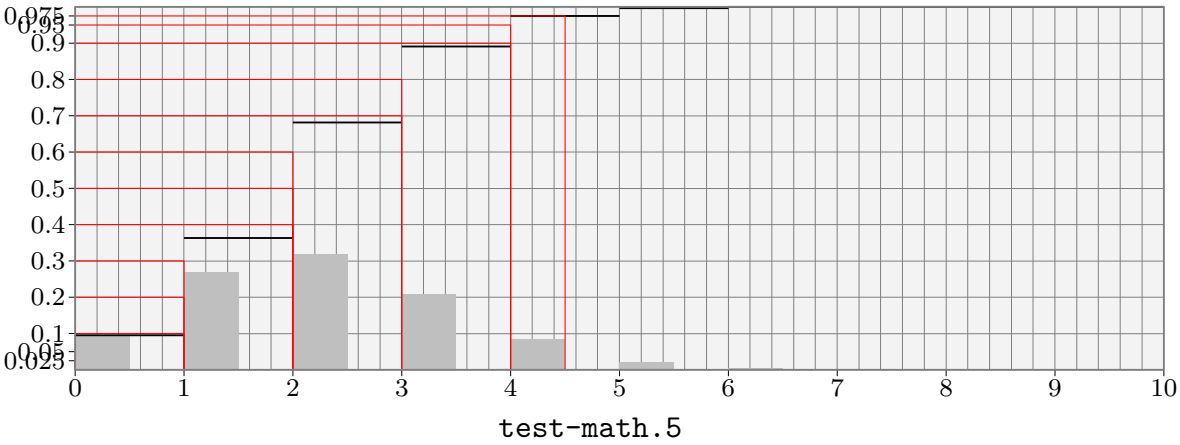
test-math.3

negativebinomial distribution with parameters (2,0.2)



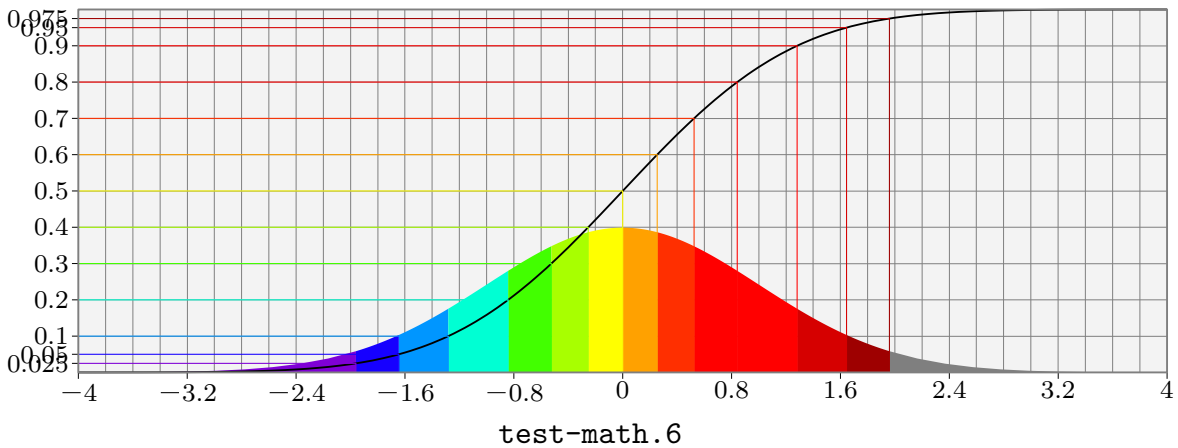
`test-math.4`

hypergeometric distribution with parameters (100,10,0.2)



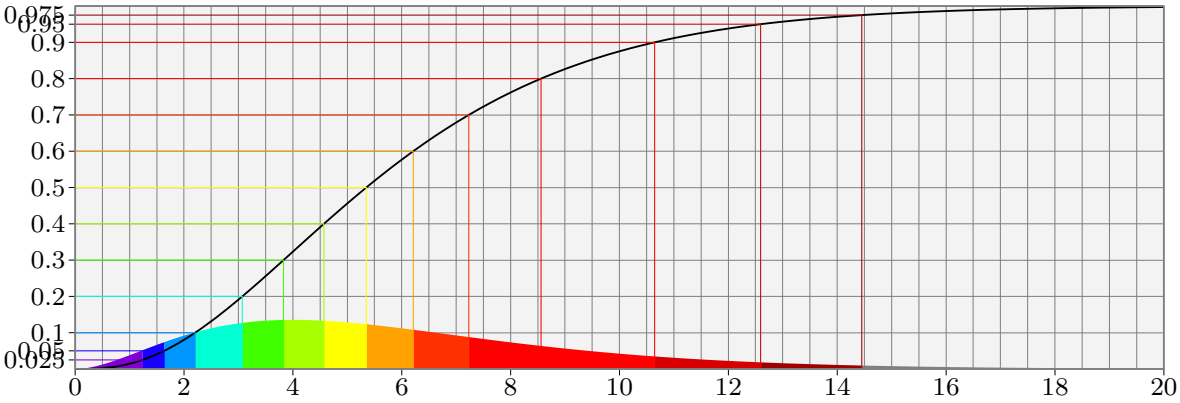
`test-math.5`

gaussian distribution with parameters (0,1)



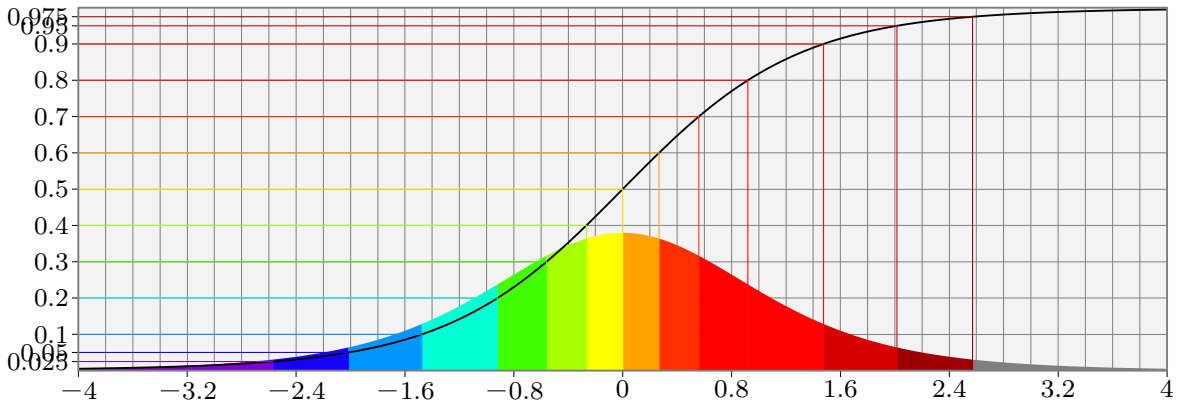
`test-math.6`

chisquare distribution with parameter 6



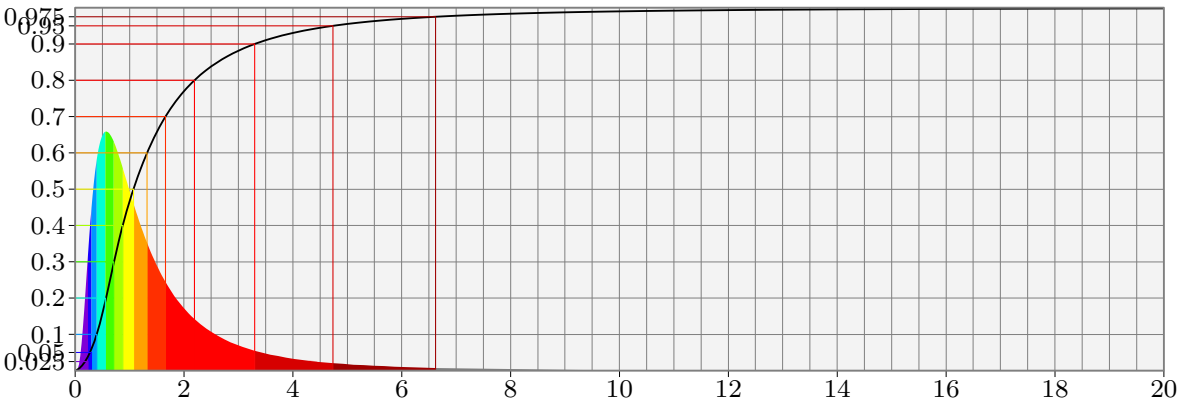
`test-math.7`

student distribution with parameter 5



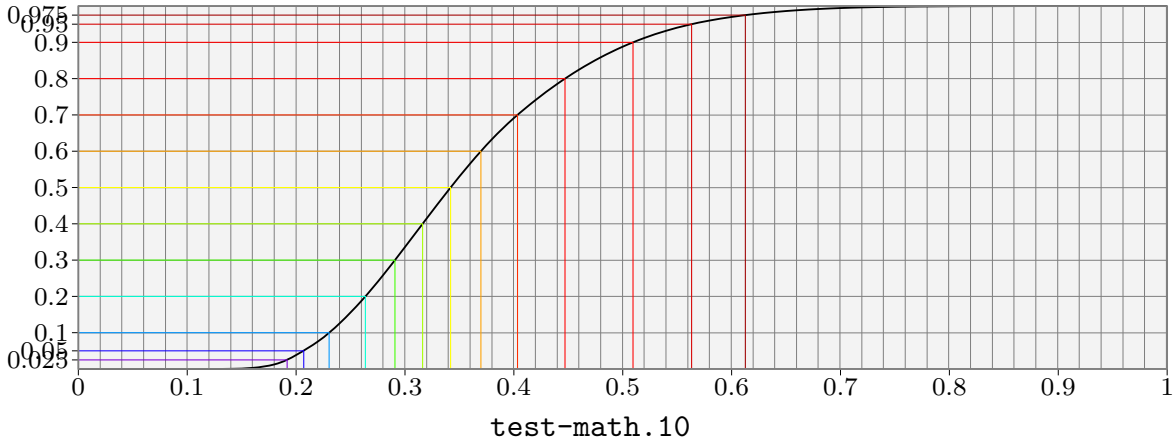
`test-math.8`

fisher distribution with parameters (10,5)



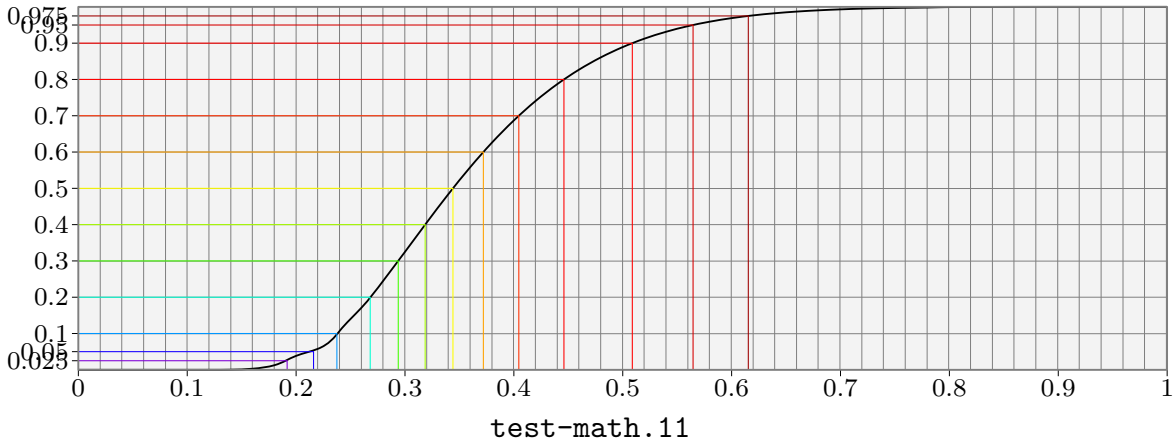
`test-math.9`

kolmogorov distribution with parameter 5



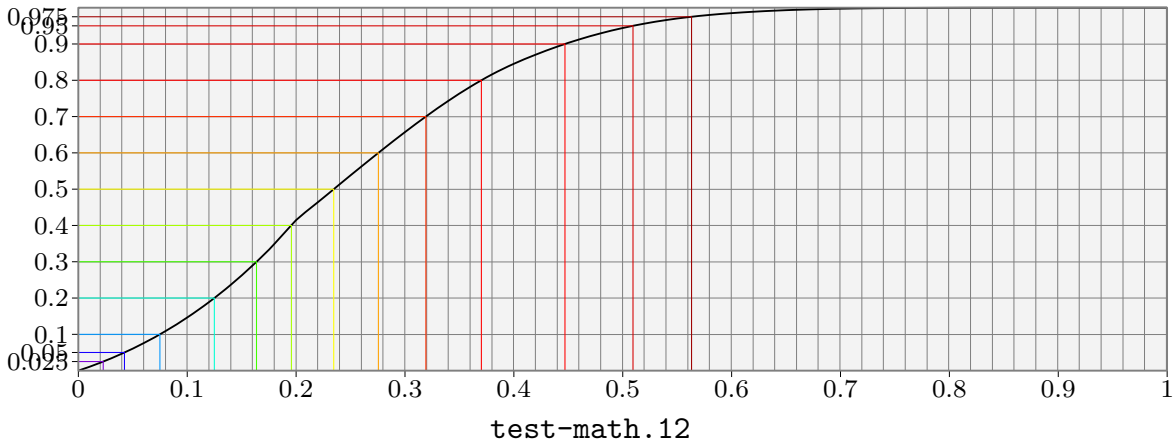
`test-math.10`

klm distribution with parameter 5



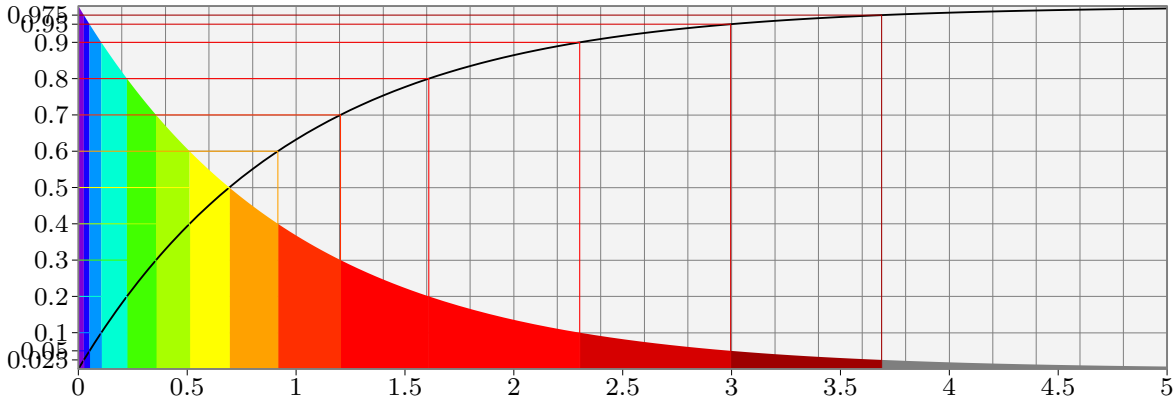
`test-math.11`

kpm distribution with parameter 5



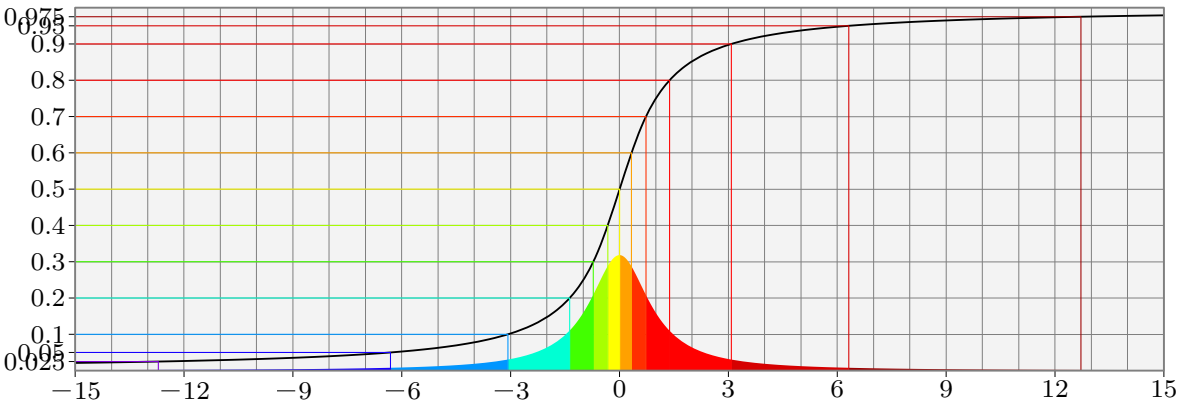
`test-math.12`

exponential distribution with parameter 1



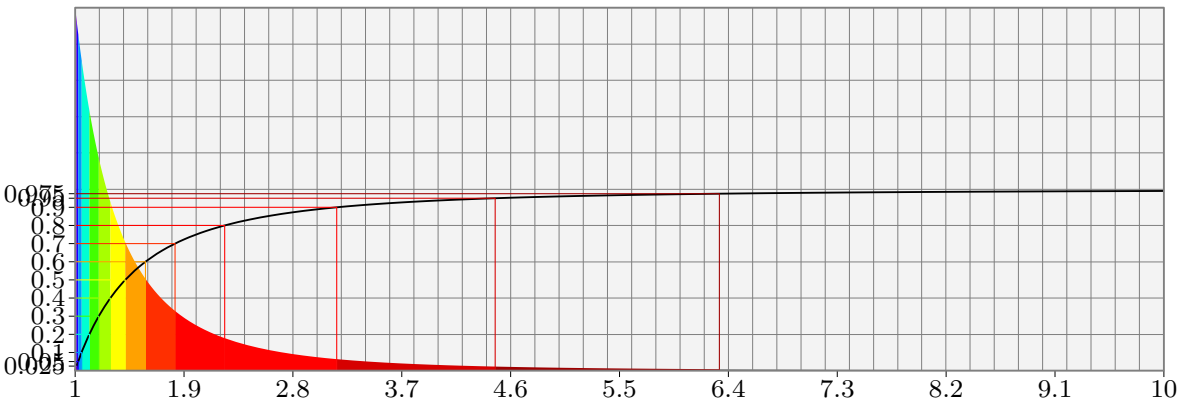
test-math.13

cauchy distribution with parameter 1



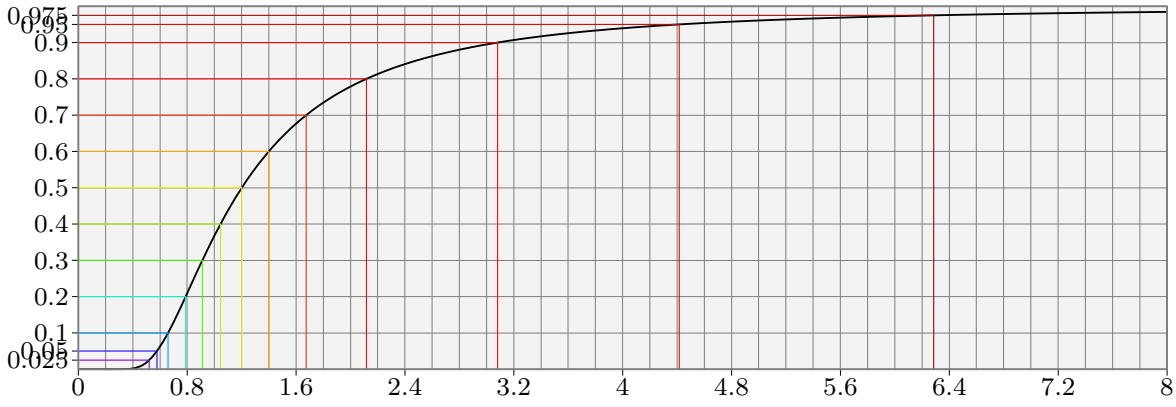
test-math.14

pareto distribution with parameters (1,2)



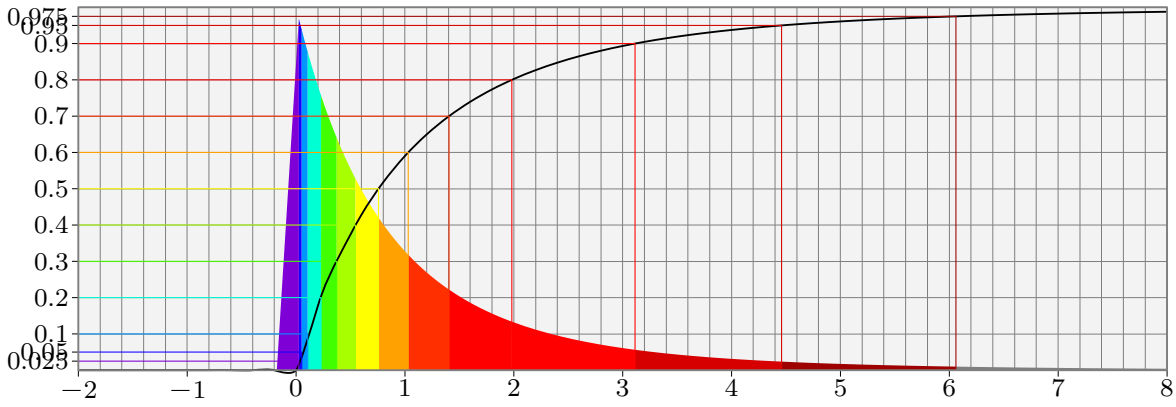
test-math.15

frechet distribution with parameters (0,1,2)



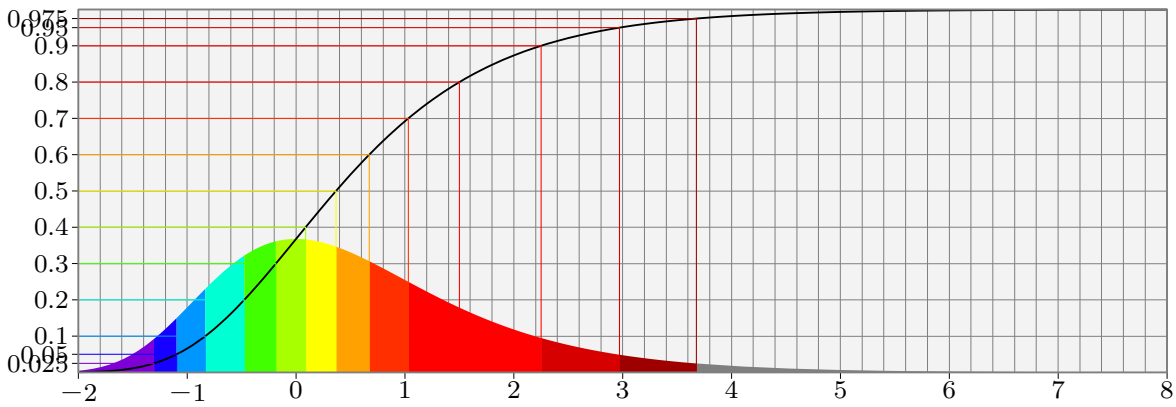
`test-math.16`

GPa distribution with parameters (0,1,0.25)

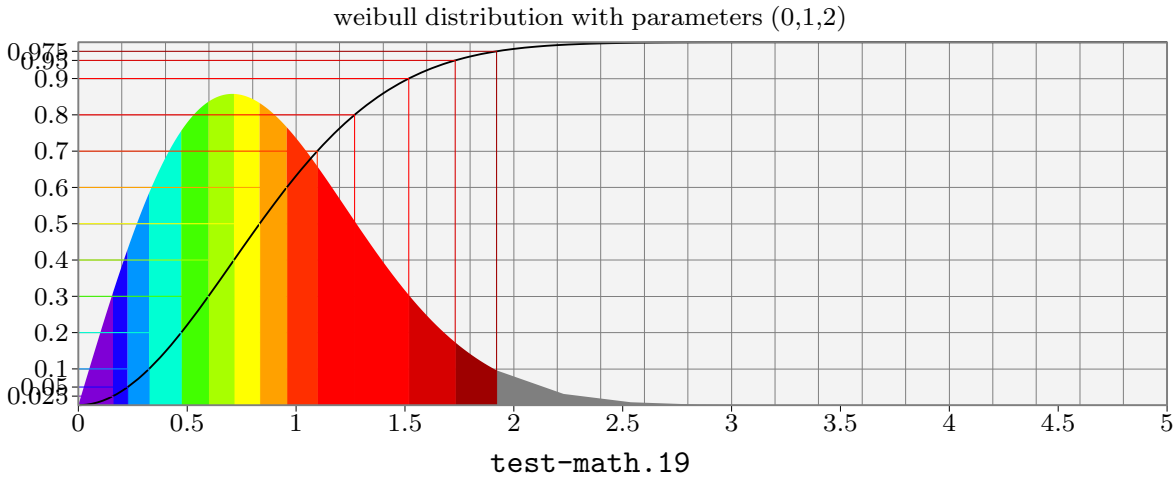


`test-math.17`

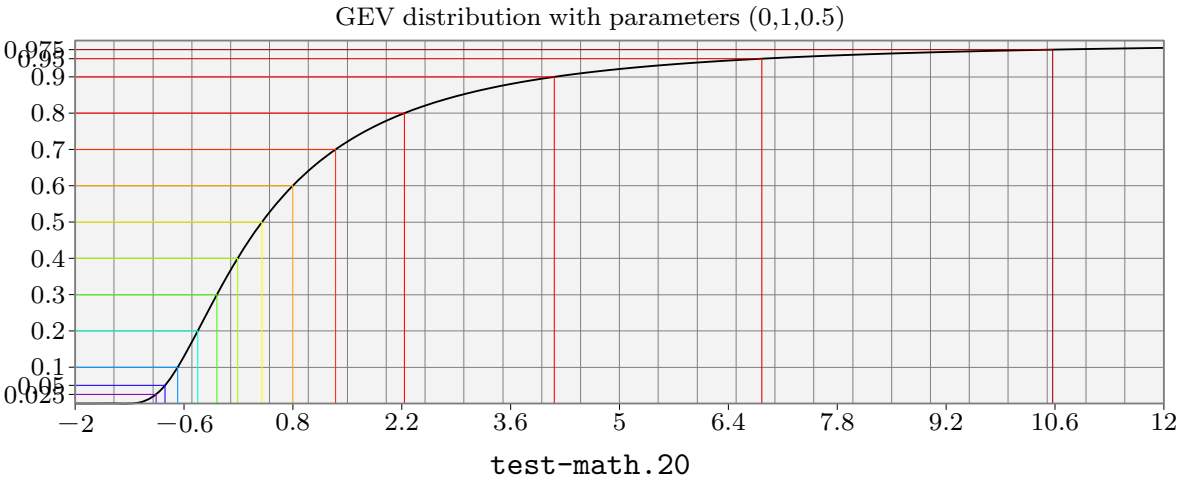
gumbel distribution with parameters (0,1)



`test-math.18`



`test-math.19`



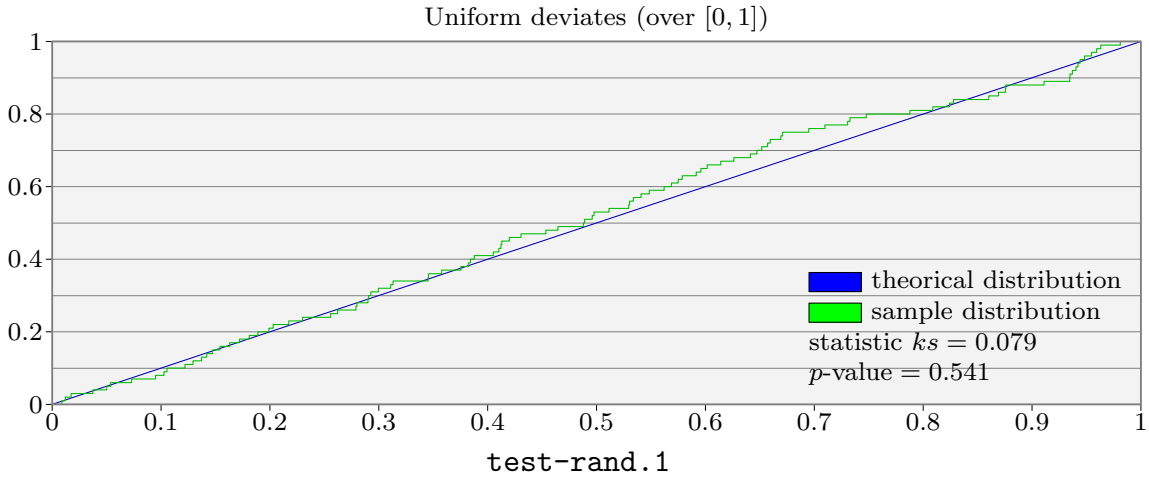
`test-math.20`

MPS-RAND.MP

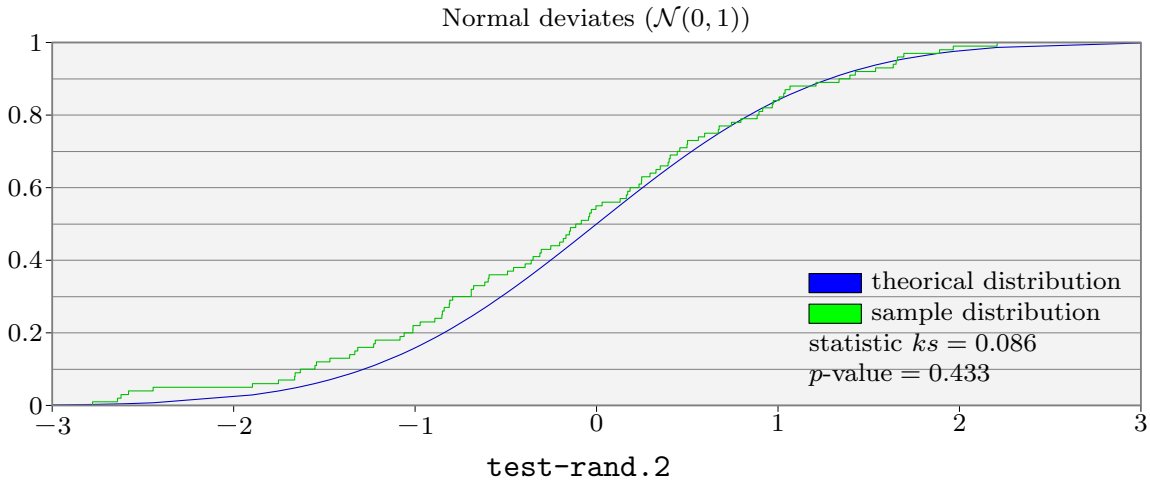
MetaPost Promotion Pages

Anthony Phan

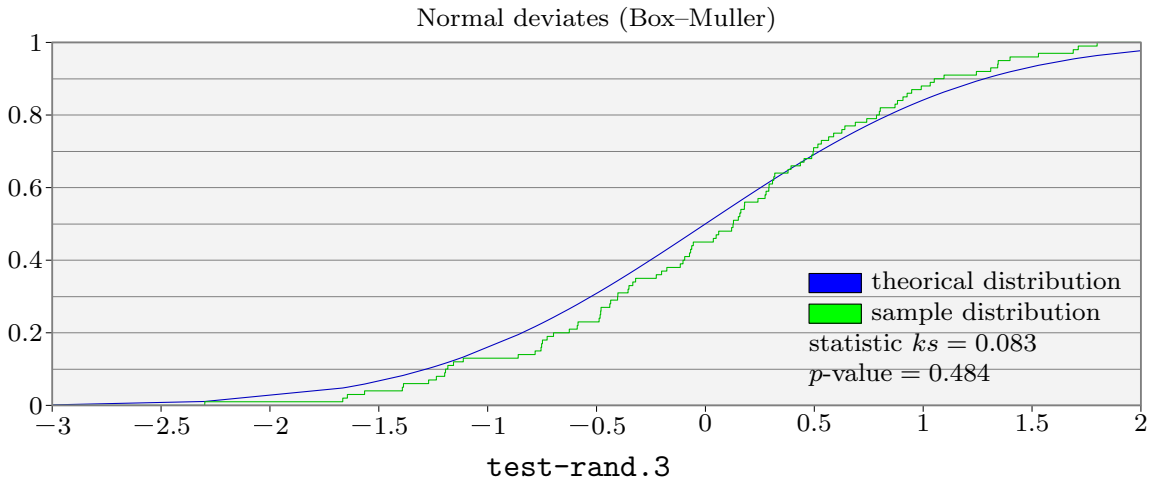
`uniformdeviate` *expr* x`.` Metafont/Post primitive.



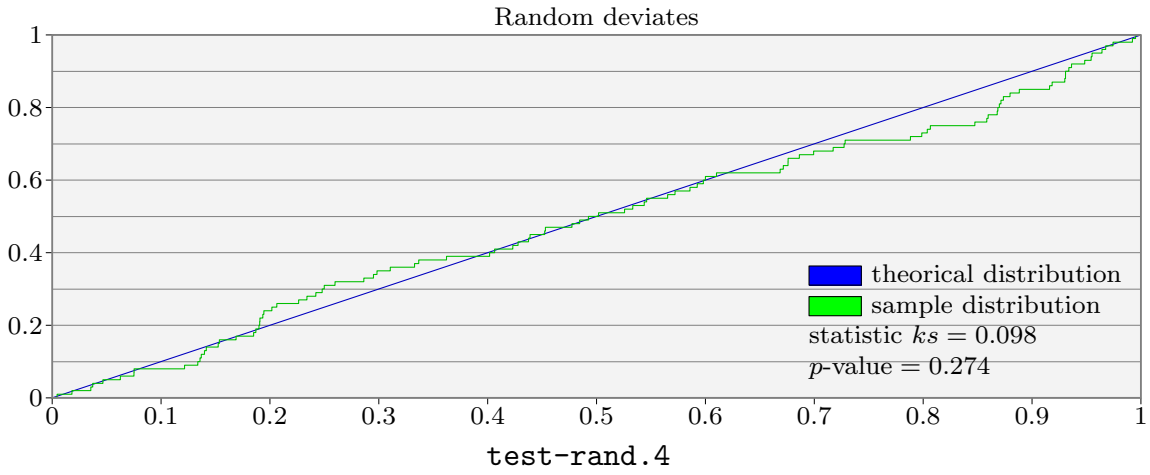
`normaldeviate.` Metafont/Post primitive.



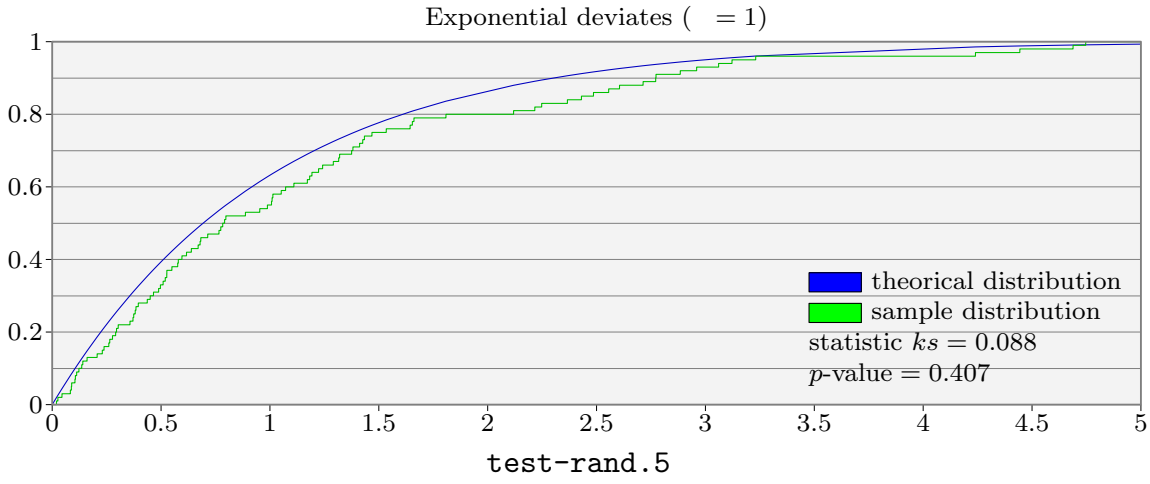
`Normaldeviate.` Box–Muller, or polar, method.



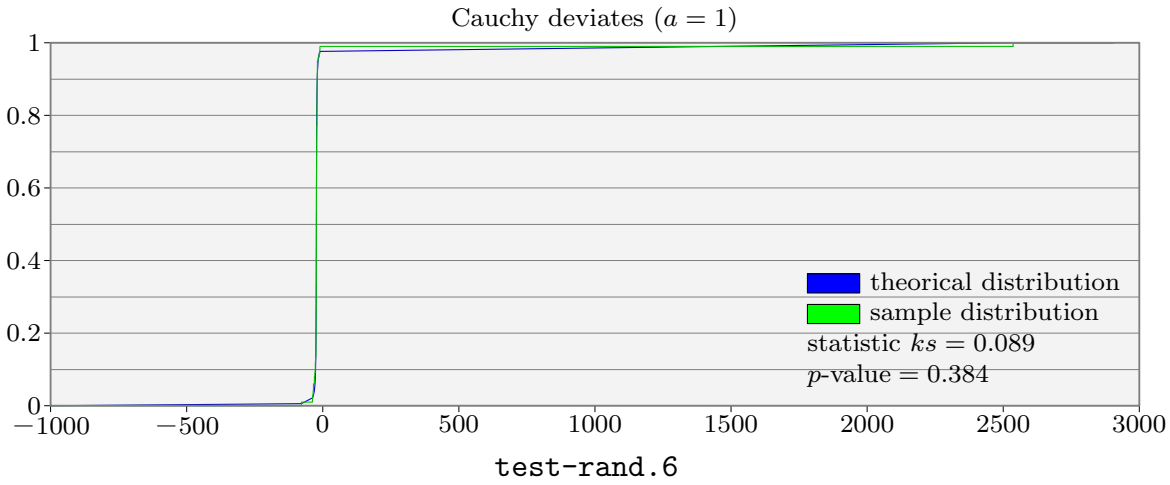
`random.`



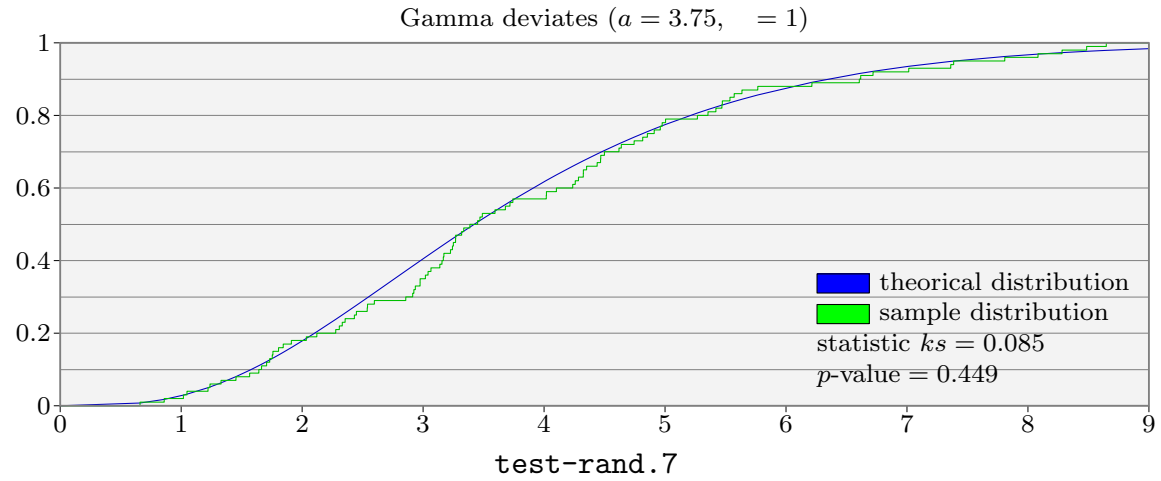
`exponentialdeviate` *expr* λ.



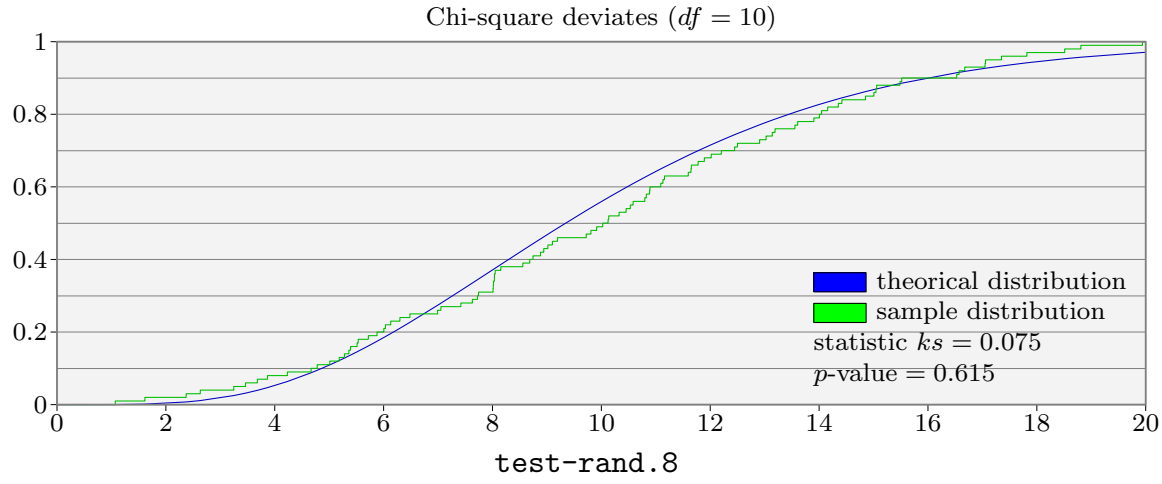
`cauchydeviate` *expr a*.



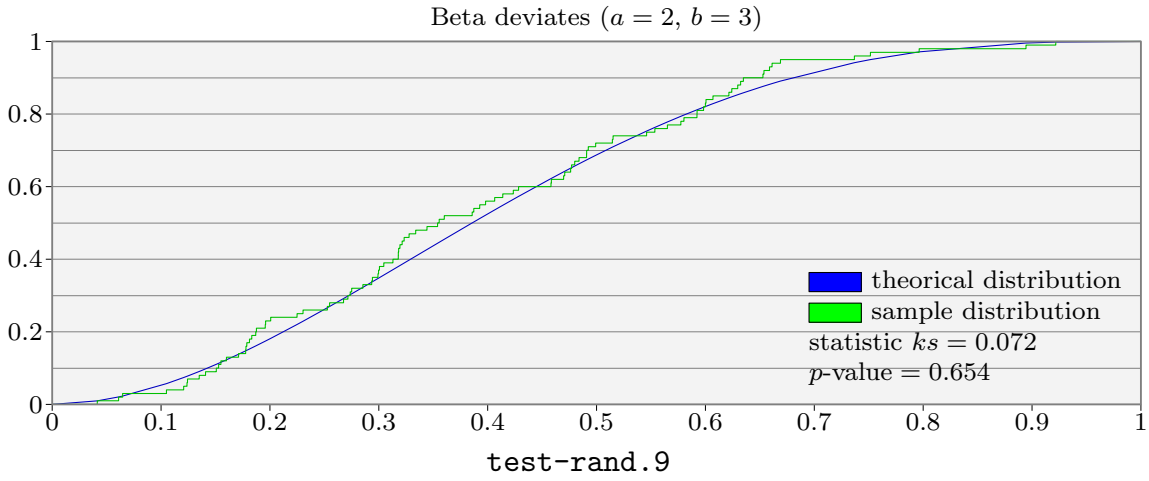
`gammadeviate(`*expr a*`, λ).`

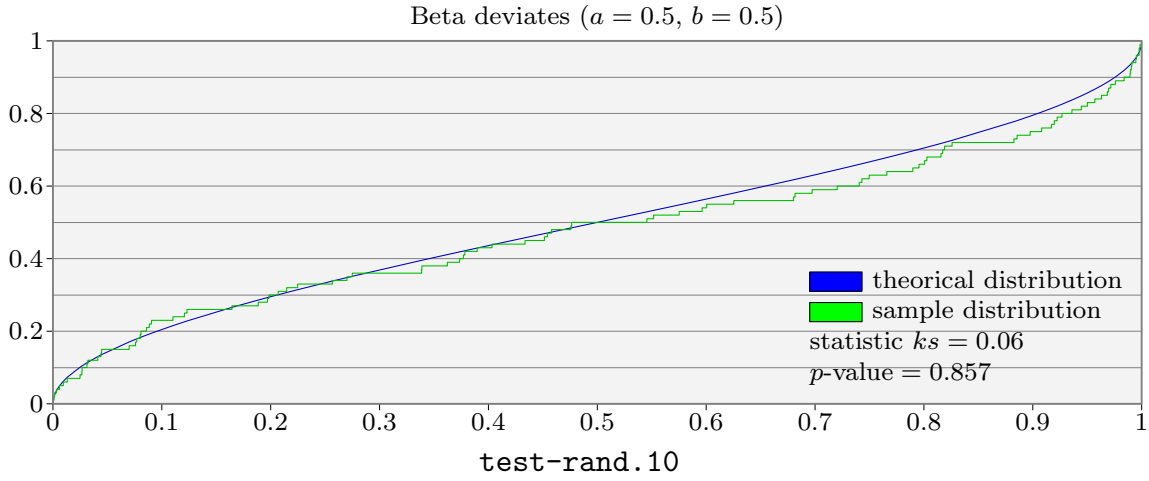


`chisquaredeviate` *expr df*.

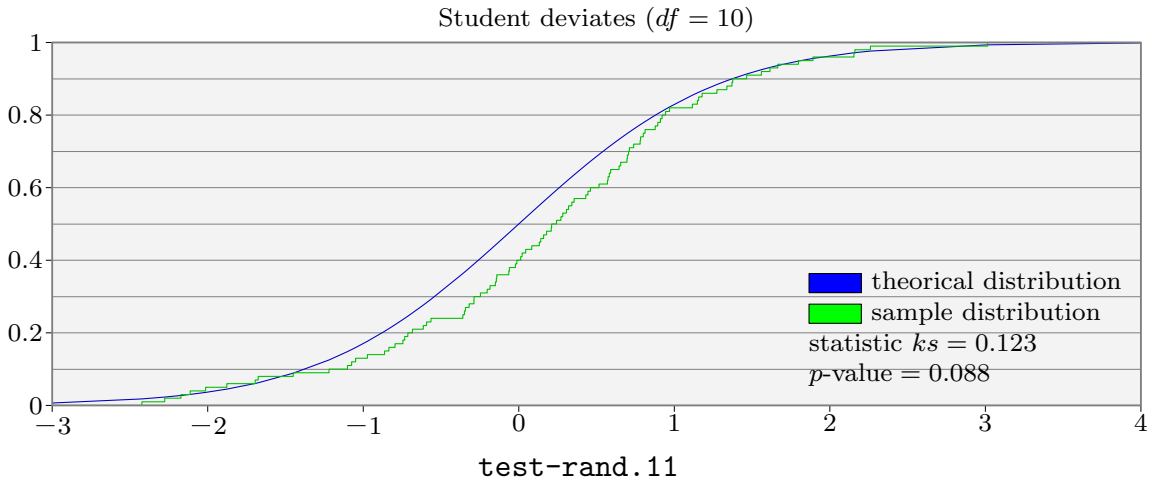


`betadeviate(`*expr* *a*`,` *b*`).`

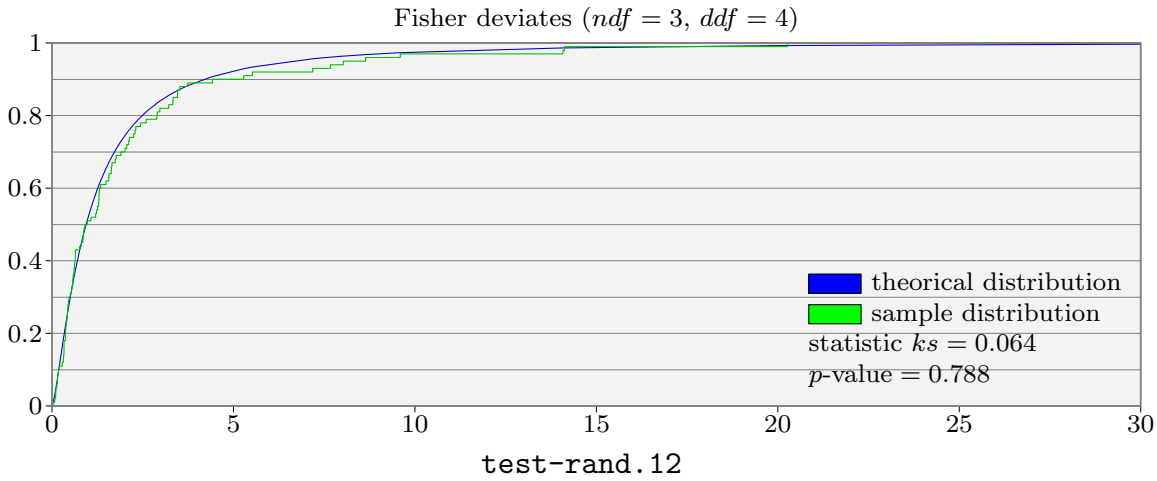




`studentdeviate` *expr* *df*`.`

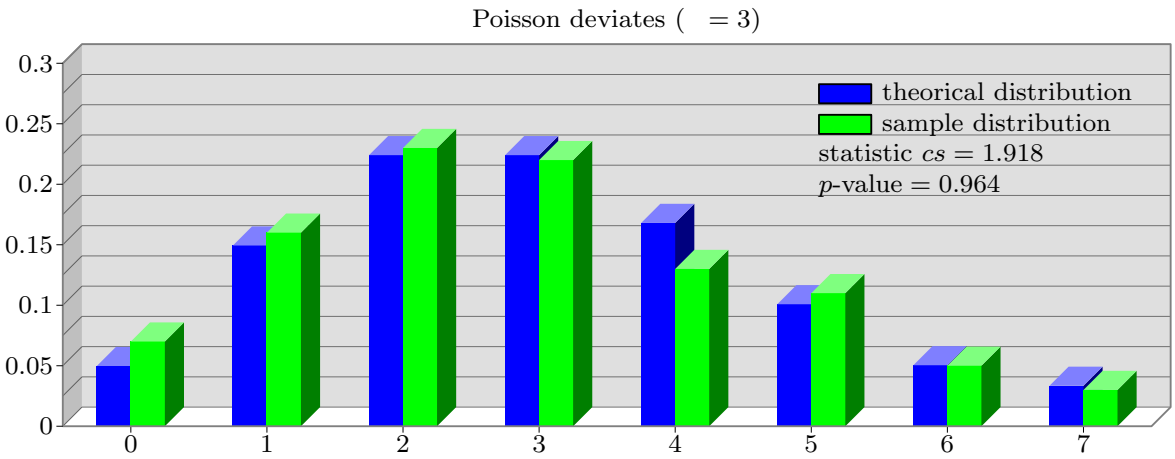


`fisherdeviate(`*expr ndf*`,` *ddf*`).`



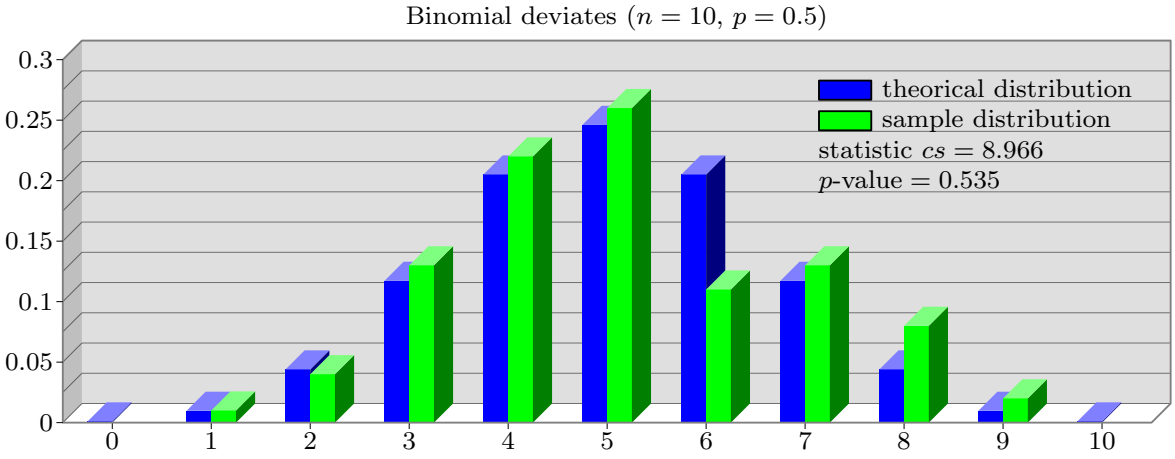
`test-rand.12`

`poissondeviate` *expr* λ`.`



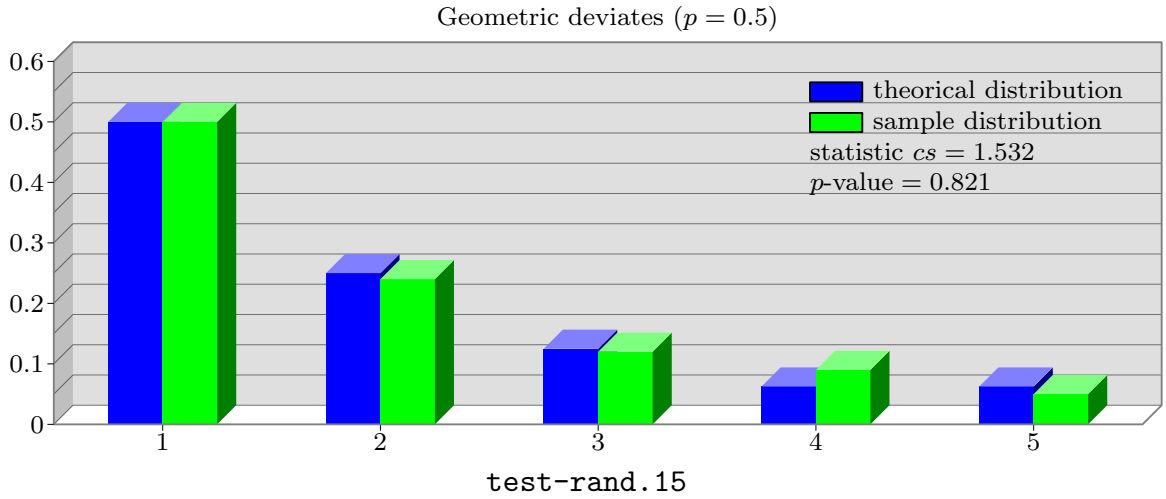
`test-rand.13`

`binomialdeviate(`*expr* n`,` π`).`



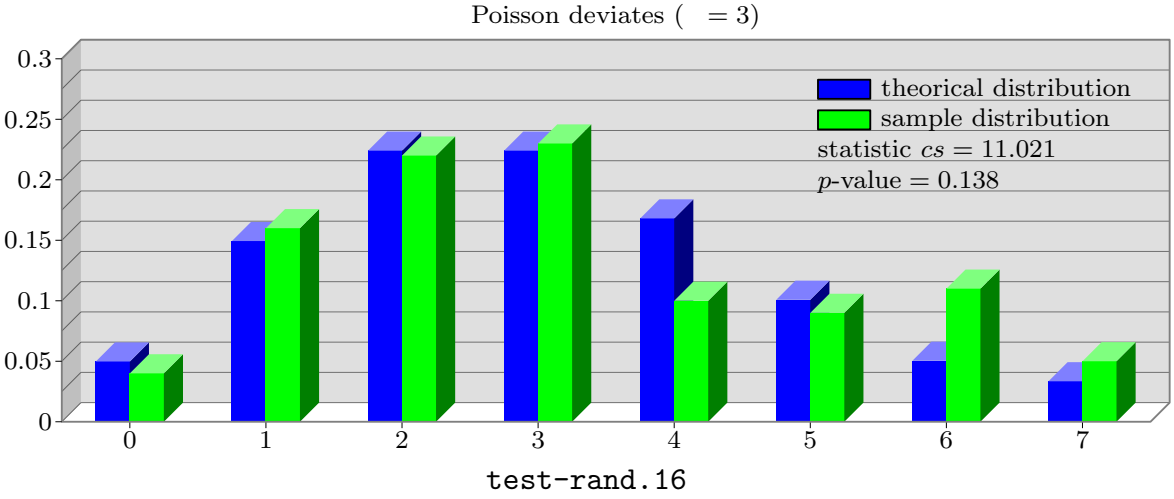
`test-rand.14`

`geometricdeviate` *expr* π.



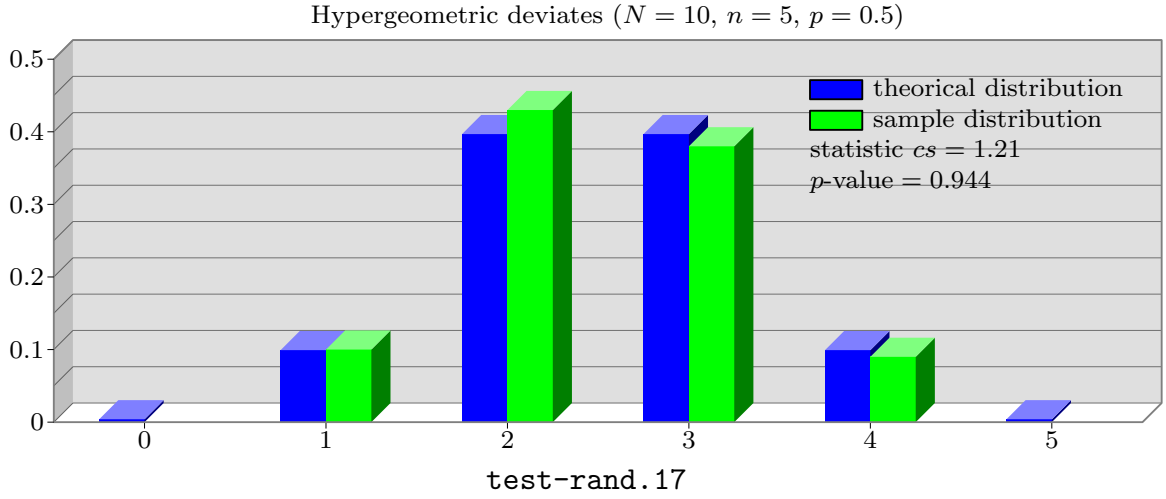
`test-rand.15`

`negativebinomialdeviate(`*expr* n`,` π`).`



`test-rand.16`

`hypergeometricdeviate(`*expr* N`,` n`,` π`).`



`test-rand.17`

`randomperm_(`*suffix* o`,` *expr* n`).` The variable o is defined to be an array $o[1], \ldots, o[n]$ which is a "random" permutation of $\{1, \ldots, n\}$. The variable o`.n` is set to be equal to n. The method is based on rejection method. Here are some possible outputs for $n = 7$.

```
{2, 7, 3, 4, 1, 5, 6}
{6, 4, 2, 1, 3, 5, 7}
{2, 6, 7, 1, 5, 4, 3}
{7, 1, 4, 3, 5, 6, 2}
{7, 3, 4, 2, 1, 5, 6}
{2, 5, 7, 4, 6, 3, 1}
{2, 5, 3, 1, 6, 4, 7}
{3, 2, 1, 5, 6, 4, 7}
{1, 7, 5, 2, 4, 6, 3}
{2, 6, 7, 4, 3, 1, 5}
```

`randomperm(`*suffix o*`,` *expr n*`)`. Same thing based on a decreasing stack. It is supposed to cost less than the previous method.

Table of Contents