# XploRe 4.0

TK, HH

August 6, 1997

# Contents

## Chapter 1

# Xplore functions

**Group:** Mathematical Functions
**Subgroup:** Common Functions
**See also:**

## Function: abs

**Description:** abs computes the absolute value of the elements of an array.

**Usage:** $z = abs(x)$

**Input:** $x$ d1 x d2 x ... x dn array

**Output:** $z$ d1 x d2 x ... x dn array

**Example:**

```
x = #(-1, 3)~#(1, -3)
abs(x)
```

**Result:**

```
[1,]     1     1
[2,]     3     3
```

**Group:** Mathematical Functions
**Subgroup:** Trigonometric Functions
**See also:** acosh, asin, asinh, atan, atan2, atanh, cos, cosh, sin, sinh, tan, tanh

## Function: acos

**Description:** acos returns the arcus cosine of the elements of an array.

**Usage:** *z = acos (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Example:**
```
x = #(0, 1/sqrt(2))~#(-1, -1/sqrt(2))
acos (x)
```

**Result:**
```
[1,] 1.5708 3.1416
[2,] 0.7854 2.3562
```

**Group:** Mathematical Functions
**Subgroup:** Trigonometric Functions
**See also:** acos, asin, asinh, atan, atan2, atanh, cos, cosh, sin, sinh, tan, tanh

## Function: acosh

**Description:** acosh returns the arc cosine hyperbolicus of the elements of an array.

**Usage:** *z = acosh (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Example:**

```
x = #(1, 2)~#(3, 4)
acosh (x)
```

**Result:**

```
[1,]     0 1.7627
[2,] 1.317 2.0634
```

**Group:** Plot Manipulation
**See also:** createdisplay, show

## Function: addbutton

**Description:** addbutton adds a button with some functionality to the plot menu.

**Usage:** *addbutton(di, l_num, c_num, name, function)*

**Input:**

| | |
|---|---|
| *di* | Display |
| *l_num* | integer, 1 ... count of lines of di |
| *c_num* | integer, 1 ... count of columns of di |
| *name* | string, contents the name for button |
| *function* | string, contents the macro, that must be proceeded as the button will be clicked |

**Example:**

```
proc () = addknn(d, i, j)
data=getdata(d, i, j, 1)
kval=readvalue("Please enter knn-parsameter:", rows(data)/10)
knnline=data[,1]~knn(data[,2], kval)
setmaskp(knnline, 0, 0, 0)
setmaskl(knnline, (1:rows(knnline))', 0, 1, 1)
adddata(d, i, j, knnline)
endp

di=createdisplay(1, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
show(di, 1, 1, x~y)
addbutton(di, 1, 1, "Add knn", "addknn(di, 1, 1)")
```

**Result:** `you see two datasets: the origin one and the knn-smoother.`

**Group:** Plot Manipulation
**See also:** deletedata, getdata, show

## Function: adddata

**Description:** adddata adds some datasets to already plotted ones.

**Usage:** *adddata(di, l_num, c_num, data1, data2, ...)*

**Input:**

| | |
|---|---|
| *di* | Display |
| *l_num* | integer, 1 ... count of lines of di |
| *c_num* | integer, 1 ... count of columns of di |
| *data1* | data matrix with min. 2 columns |

**Example:**

```
di=createdisplay(1, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
show(di, 1, 1, x~y)
y=knn(y, 10)
adddata(di, 1, 1, x~y)
connect(di, 1, 1, 2, 4, 3)
```

**Result:** `you see two datasets: the origin one and the knn-smoother.`

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** list, insert, delete, names, comp, dot, size, {}

## Function: append

**Description:** appends an object to the specified list. If the appended object is temporary, the component gets the name el<position>. If the first argument is not a composed object, it will be the first component of the resulting list.

**Usage:** *append (l, x)*

**Input:**

| | |
|---|---|
| *l* | list |
| *x* | object |

**Notes:** Although it is allowed that a list can contain components with the same name, it is not recommended.

**Example:**

```
l = list (1, 2)
append (l, 100)
l
```

**Result:**

```
Content of object l.el1

[1,]        1
Content of object l.el2

[1,]        2
Content of object l.el3

[1,]      100
```

**Group:** Mathematical Functions
**Subgroup:** Trigonometric Functions
**See also:** acos, acosh, asinh, atan, atan2, atanh, cos, cosh, sin, sinh, tan, tanh

## Function: asin

**Description:** asin returns the arcus sine of the elements of an array.

**Usage:** $z = asin(x)$

**Input:** $x$ d1 x d2 x ... x dn array

**Output:** $z$ d1 x d2 x ... x dn array

**Example:**

```
x = #(1, 1/sqrt(2))~#(0, 1/sqrt(2))
asin(x)
```

**Result:**

```
[1,] 1.5708      0
[2,] 0.7854 0.7854
```

**Group:** Mathematical Functions
**Subgroup:** Trigonometric Functions
**See also:** acos, acosh, asin, atan, atan2, atanh, cos, cosh, sin, sinh, tan, tanh

## Function: asinh

**Description:** asinh returns the arc sine hyperbolicus of the elements of an array.

**Usage:** $z = asinh(x)$

**Input:** $x$ d1 x d2 x ... x dn array

**Output:** $z$ d1 x d2 x ... x dn array

**Example:**
```
x = #(0, 2)~#(3, 4)
asinh(x)
```

**Result:**
```
[1,]      0 1.8184
[2,] 1.4436 2.0947
```

**Group:** Mathematical Functions
**Subgroup:** Trigonometric Functions
**See also:** acos, acosh, asin, asinh, atan2, atanh, cos, cosh, sin, sinh, tan, tanh

**Function:** **atan**

**Description:** atan returns the arctangus in radian of the elements of an array.

**Usage:** $z = atan(x)$

**Input:** $x$ d1 x d2 x ... x dn array

**Output:** $z$ d1 x d2 x ... x dn array

**Example:**

```
x = #(sqrt(3), 1)~#(0, -1)
atan(x)
```

**Result:**

```
[1,] 1.0472      0
[2,] 0.7854 -0.7854
```

**Group:** Mathematical Functions
**Subgroup:** Trigonometric Functions
**See also:** acos, acosh, asin, asinh, atan, atanh, cos, cosh, sin, sinh, tan, tanh

## Function: atan2

**Description:** atan2 returns the angle in radian between the positive part of the x-axis and the line with origin in (0,0) which contains the point (x, y).

**Usage:** *z = atan2(y, x)*

| | | |
|---|---|---|
| **Input:** | *x* | d1 x d2 x ... x dn array |
| | *y* | d1 x d2 x ... x dn array |
| **Output:** | *z* | d1 x d2 x ... x dn array |

**Example:** `atan2(0, -1.0)`

**Result:** `[1,] 3.1416`

**Group:** Mathematical Functions
**Subgroup:** Trigonometric Functions
**See also:** acos, acosh, asin, asinh, atan, atan2, cos, cosh, sin, sinh, tan, tanh

## Function: atanh

**Description:** atanh returns the hiperbolic arc tangent of the elements of an array.

**Usage:** *z = atanh(x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Notes:** the absolute value of the input values must be $< 1$. otherwise NaN's are returned.

**Example:**
```
x = #(0, 1/2)~#(1/3, 1/4)
atanh(x)
```

**Result:**
```
[1,]       0 0.34657
[2,] 0.54931 0.25541
```

**Group:**
**See also:** string substr

## Function: atof

**Description:** atof is a command to convert strings in numbers.

**Usage:** $z$ = *atof(x)*

**Input:** $x$ array

**Output:** $z$ array

**Notes:** If the string does not contain a number the result follows the definiton of the C-function atof.

**Example:**
```
s = #("1", "2")~#("3", "4")
z = atof(s)
z
```

**Result:**
```
Content of object z

[1,]        1       3
[2,]        2       4
```

**Group:** Plot Manipulation
**See also:** createdisplay, adddata, deletedata, getdata, show, axeson

## Function: axesoff

**Description:** all the plots created later become unvisible axis

**Usage:** *axesoff()*

**Example:**
```
axeson()
di=createdisplay(2, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
show(di, 1, 1, x~y)
axesoff()
show(di, 2, 1, x~y)
```

**Result:** `only the upper plot has visible axis`

**Group:** Plot Manipulation
**See also:** createdisplay, adddata, deletedata, getdata, show, axesoff

**Function:** **axeson**

**Description:** all the plots created later become visible axis

**Usage:** *axeson()*

**Example:**
```
axeson()
di=createdisplay(2, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
show(di, 1, 1, x~y)
axesoff()
show(di, 2, 1, x~y)
```

**Result:** `only the upper plot has visible axis`

**Group:** Optimizer
**See also:** nelmin

## Function: bfgs

**Description:** bfgs searchs the global minimum of a function.

**Usage:** *x = bfgs (x0, f, df, maxiter {,eps, gamma})*

**Input:**

| | |
|---|---|
| *x0* | n x 1 vector of starting value |
| *f* | text, name of the procedure where the function is defined |
| *df* | text, name of the procedure where the function derivative is defined |
| *maxiter* | integer, maximal number of iterations |
| *eps* | scalar, bfgs stops, if df(x_i) < eps (i=1...n) (default = 1.0e-03) |
| *gamma* | scalar, Startvalue for linesearch (default = 1.0) |

**Output:**

| | |
|---|---|
| *x.minimum* | n x 1 vector with found minimum coordinates |
| *x.iter* | number of iterations |
| *x.converged* | 1 if the algorithm has converged and 0 if it has not |

**Example:**

```
proc (y) = f(x)
y = sum((x-2)^2)
endp
proc (y) = df (x)
y = 2*(x-2)
endp

x0=#(1,1,1)
bfgs(x0,"f","df",50)
```

**Result:**

```
Content of object bfgs.minimum

[1,]     2
[2,]     2
[3,]     2
Content of object bfgs.iter

[1,]    50
Content of object bfgs.converged

[1,]     0
```

**References:**

Bulirsch, R. and Stoer, J. (1980). "Introduction to Numerical Analysis", p. 302, Springer-Verlag.

Schwetlick, H. (1979). "Numerische Loesung Nichtlinearer Gleichungen", p.187, VEB Deutscher Verlag Wissenschaften.

**Group:** Binning, Grids and Sequences
**See also:** aseq, conv, distance, grid, mseq

## Function: bindata

**Description:** This command bins a p-dimensional data set starting from origin or in steps of d.

**Usage:** *bindata (x, d, or {,y})*

**Input:**

| | | |
|---|---|---|
| | *x* | m x n x d1 x ... x dn array |
| | *d* | n x 1 x d1 x ... x dn array |
| | *or* | n x 1 x d1 x ... x dn array |
| | *y* | m x p x d1 x ... x dn array |

**Output:**

| | | |
|---|---|---|
| | *xb* | l x n x d1 x ... x dn array |
| | *yb* | l x (p+1) x d1 x ... x dn array |

**Notes:** If you do not include a fourth argument x is taken instead of y.

**Example:**

```
x=aseq(0,10,0.1)
y=aseq(2,10,0.2)
z=x~y
d=#(0.4,0.8)
org=#(0.05,1.95)
{xa, xb} = bindata(z,d,org)
xa
xb
```

**Result:**

```
Content of object bindata.xb
-1     0
0     0
0     0
0     0
0     1
1     1
1     1
1     1
1     2
2     2
Content of object bindata.yb

1
3
0
0
1
3
0
0
1
1
```

**Remarks:** Bins a p-dimensional data set x (a matrix n x p) starting from origin org in steps of d. The binning parameter d can be a scalar or a p-vector which gives a specific step for each column xj (j=1,...,p).

The output matrix xb contains the index of the nonempty bins.
The output matrix yb contains in the first column the abso- lute frequency of datapoints xij in each nonempty bin.

If the optional parameter y (a n x L matrix) is given then the sum of the yi's (i=1,...,n) such that the corresponding xij lies in a bin is given back respectively on the last L columns of yb.
.

**Group:** Flow Control
**See also:** switch case default endsw

## Function: break

**Description:** Inside a switch statement break marks the end of a case block. The procedure is continued at the keyword endsw. It can be omitted.

**Usage:** *switch case (cond) break default endsw*

**Notes:** Even if the condition is a multidimensional array only the first element is checked for TRUE (!=0) or FALSE (==0)

**Example:**
```
proc (s) = sign (x)
switch
case (x<0)  y = -1  break
case (x>0)  y =  1  break
default     y =  0  break   ; this break has no effect
endsw
endp
```

**Group:** Flow Control
**See also:** switch break default endsw

**Function:** **case**

**Description:** Inside a switch-endsw block case controls the execution of an alternative. If the condition of case is true, the following block is executed similar to an if-endif statement. The keyword break serves as end marker of case and leaves the switch block at the position of endsw. When break is omitted, the next consecutive case is processed. If the program's counter comes to default, the following block is executed in any case.

**Usage:** *switch case (cond) break default endsw*

**Notes:** Even if the condition is a multidimensional array only the first element is checked for TRUE (!=0) or FALSE (==0)

**Example:**
```
proc (s) = sign (x)
switch
case (x<0)  y = -1  break
case (x>0)  y =  1  break
default     y =  0  break   ; this break has no effect
endsw
endp
```

**Group:** Statistical Distributions
**See also:** cdfc, cdff, cdfn, cdft, pdfb, pdfc, pdff, pdfn, pdft, qfc, qff, qfn, qft

## Function: cdfb

**Description:** cdfb returns the beta-distribution function for elements of an array.

**Usage:** *z = cdfb (x, a, b)*

| | | |
|---|---|---|
| **Input:** | *x* | n x p x d1 x d2 x ... x dn array |
| | a | n x p x d1 x d2 x ... x dn array or scalar |
| | *b* | n x p x d1 x d2 x ... x dn array or scalar |
| **Output:** | *z* | n x p x d1 x d2 x ... x dn array |

**Notes:** cdfb is only defined for values 0<=x<=1, a>0, b>0, otherwise missing values are generated.

**Example:** `cdfb (0.5, 1.0, 1.0 )`

**Result:**

```
0.49998
```

**Group:** Statistical Distributions
**See also:** cdfb, cdff, cdfn, cdft, pdfb, pdfc, pdff, pdfn, pdft, qfc, qff, qfn, qft

## Function: cdfc

**Description:** cdfc returns the chi-quare distribution function with d degrees of freedom for elements of an array.

**Usage:** *z = cdfc (x,d)*

| | | |
|---|---|---|
| **Input:** | *x* | n x p x d1 x d2 x ... x dn array |
| | *d* | n x p x d1 x d2 x ... x dn array or scalar |
| **Output:** | *z* | n x p x d1 x d2 x ... x dn array |

**Notes:** For elements xij $<$ 0 missing values are generated.

**Example:** `cdfc(12.59, 6)`

**Result:** `0.95`

**Group:** Statistical Distributions
**See also:** cdfb, cdfc, cdfn, cdft, pdfb, pdfc, pdff, pdfn, pdft, qfc, qff, qfn, qft

## Function: cdff

**Description:** cdff returns the F-distribution function with d1 and d2 degrees of freedom for elements of an array.

**Usage:** *z = cdff (x, d1, d2)*

| | | |
|---|---|---|
| **Input:** | *x* | n x p x d1 x d2 x ... x dn array |
| | *d1* | n x p x d1 x d2 x ... x dn array or scalar |
| | *d2* | n x p x d1 x d2 x ... x dn array or scalar |
| **Output:** | *z* | n x p x d1 x d2 x ... x dn array |

**Notes:** For elements xij < 0 missing values are generated.

**Example:** `cdff (5.41,3, 5 )`

**Result:** `0.95001`

**Group:** Statistical Distributions
**See also:** cdfb, cdfc, cdff, cdft, normal, pdfb, pdfc, pdff, pdfn, pdft, qfc, qff, qfn, qft

## Function: cdfn

**Description:** cdfn returns the standard normal distribution function for elements of an array.

**Usage:** *z = cdfn (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Example:** `cdfn (1.96)`

**Result:** `[1,] 0.975`

**Group:** Statistical Distributions
**See also:** cdfb, cdfc, cdff, cdfn, pdfb, pdfc, pdff, pdfn, pdft, qfc, qff, qfn, qft

## Function: cdft

**Description:** cdft returns the t-distribution function with d degrees of freedom for elements of an array.

**Usage:** *z = cdft(x,d)*

**Input:** *x* n x p x d1 x d2 x ... x dn array
*p* n x p x d1 x d2 x ... x dn array or scalar

**Output:** *z* n x p x d1 x d2 x ... x dn array

**Example:** `cdft (2.92, 16)`

**Result:** `0.99499`

**Group:** Mathematical Functions
**Subgroup:** Common Functions
**See also:** floor

## Function: ceil

**Description:** ceil gives the next highest integer value of the elements of an array.

**Usage:** $z$ = *ceil* ($x$)

**Input:** $x$ d1 x d2 x .. x dn array

**Output:** $z$ d1 x d2 x .. x dn array

**Example:**

```
x = #(0.79, 2.1) ~ #(-3.99, -6.01)
ceil (x)
```

**Result:**

```
[1,]     1    -3
[2,]     3    -6
```

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** dim, rows

**Function:** **cols**

**Description:** cols returns the number of columns in an array.

**Usage:** $z = cols\ (x)$

**Input:** $x$ d1 x d2 x ... x dn array

**Output:** $z$ scalar with the number d2

**Example:** `cols (matrix (10, 3))`

**Result:** `[1,]     3`

**Group:** Information and Control
**See also:** list, size

## Function: comp

**Description:** comp checks, whether an object has a specific component. If the first argument is a name, the object with the specified name is used as list object.

**Usage:** *x = comp(obj, com)*

**Input:** *obj* list object or string
*com* string, containing the name of the component

**Output:** *x* scalar

**Notes:** comp can have the following results:
-1 ... object with name obj does not exist
0 ..... object has no component with name com
>0 ... position of the component in the object obj

**Example:**

```
proc () = plot (z, d)
if (comp (z, "x") && comp (z, "y"))
show (d, 1, 1, z.x~z.y)
endif
endp

d = createdisplay (1,1)
x = (1:100)/100
y = x^2
a = list (x, y)
plot (a, d)
```

**Result:** `a window with a plot of x^2`

**Group:** Plot Manipulation
**See also:**

**Function:** **connect**

**Description:** connect all points of a dataset that is already plotted.

**Usage:** *connect(di, l_num, c_num, data_number, color_number, art_number)*

**Input:**

| | |
|---|---|
| *di* | Display |
| *l_num* | integer, 1 ... count of lines of di |
| *c_num* | integer, 1 ... count of columns of di |
| *data_number* | integer, 1 ... count of dataparts of di |
| *color_number* | integer, 0 ... 7 |
| *art_number* | integer, 0 ... ... |

**Output:**

**Example:**

```
di=createdisplay(1, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
show(di, 1, 1, x~y)
y=knn(y, 10)
adddata(di, 1, 1, x~y)
connect(di, 1, 1, 2, 4, 3)
```

**Result:** `you see two datasets: the origin one and the knn-smoother. The smoother is a red thick line.`

**Group:**
**See also:** contour3

## Function: contour2

**Description:** contour2 computes lines and points for a contourplot of a three dimensional dataset at level c

**Usage:** *{lt, pt} = contour2 (x, c)*

**Input:**

| | |
|---|---|
| *x* | n x 3 matrix |
| *c* | scalar |

**Output:**

| | |
|---|---|
| *lt* | m1 x 2 line matrix |
| *pt* | m2 x 2 point matrix |

**Notes:** It is important that the data matrix x is sorted by its columns.

**Example:**

```
x = grid(-#(3,3), #(0.25,0.25), #(25,25))
f = exp(-0.5.*sum(x^2,2))./(2.*pi)
z = sort(x~f, #(1,2,3))
{lt,pt} = contour2(z, max(f)./2)
d = createdisplay(1,1)
show (d,1,1,pt)
```

**Result:** `You will see a circle corresponding to a contourline.`

**Group:**
**See also:** contour2

## Function: contour3

**Description:** contour3 computes lines and points for a contourplot of a four dimensional dataset at level c

**Usage:** *{lt, pt} = contour3 (x, c)*

**Input:**

| | | |
|---|---|---|
| | *x* | n x 4 matrix |
| | *c* | scalar |

**Output:**

| | | |
|---|---|---|
| | *lt* | m1 x 2 line matrix |
| | *pt* | m2 x 3 point matrix |

**Notes:** It is important that the data matrix x is sorted by its columns.

**Example:**

```
x = grid(-#(3,3,3), #(0.5,0.5,0.5), #(13,13,13))
f = exp(-0.5.*sum(x^2,2))./(2.*pi)^1.5
z = sort(x~f, #(1,2,3,4))
{lt,pt} = contour3(z, max(f)./2)
d = createdisplay(1,1)
show (d,1,1,pt)
```

**Result:** `You will see a ball corresponding to a contourball.`

**Group:** Binning, Grids and Sequences
**See also:** bindata, denest, grid

## Function: conv

**Description:** conv performs the convolution of a step kernel function and a function over a p-dimensional equidistant grid.

**Usage:** *{xc, yc, fill} = conv (xb, yb, wx, wy {,sym})*

**Input:**

| | |
|---|---|
| *xb* | m x p x d1 x ... x dn array of integers |
| *yb* | m x l x d1 x ... x dn array of non-negative integers |
| *wx* | r x p x d1 x ... x dn array of non-negative integers |
| *wy* | r x 1x d1 x ... x dn array |
| *sym* | scalar |

**Output:**

| | |
|---|---|
| *xc* | k x p x d1 x ... x dn array (k >= m) |
| *yc* | k x l x d1 x ... x dn array |
| *fill* | k x 1 x d1 x ... x dn array of zeros or ones |

**Notes:** conv is specially designed to calculate Kernel estimates through binning.
For p has to hold: p <= 15.

**Example:**

```
x = normal (100)
{xb,yb} = bindata(x, 0.4,0)
wx = # (0,1)
wy = # (1.25,0.75)
{xc, yc, fill} = conv (xb, yb, wx, wy)
cc=createdisplay(1,1)
show (cc,1,1,xc ~ yc)
```

**Result:** `Display of a one-dimensional kernel estimate of the normal distribution.`

**Example:**

```
randomize(0)
x = normal (100,2)
{xb,yb} = bindata(x, #(0.25,0.25),#(0,0))
library("kernel")
x = #(0,0)
h = #(0.25,0.25)
n = #(5,5)
wx= grid(x,#(1,1),n)
wy = qua(wx.*h)
{xc, yc, fill} = conv (xb, yb, wx, wy)
cc=createdisplay(1,1)
show (cc,1,1,xc ~ yc)
```

**Result:** `Display of a two-dimensional kernel estimate of the two-dimensional normal distribution.`

**Remarks:** IDEA:

conv is a computational command to improve the speed with which the kernel estimates are carried out.

In a first step you need to bin your data, e.g. with bindata.
In a second step you descreticize the kernel that you have chosen.
Both grids must have the same size.

conv performs the convolution of given values of a symmetric p-dimensional (step) function K(x), with the columns of a matrix yb with values given over a p-dimensional grid, as produced when binning your data.

INPUTS:
xb, a m x p matrix of integers with the indices of the bins in the grid (e.g., like xb as returned by bindata) and
yb, a m x 1 matrix (e.g., yb as returned by bindata),which gives you the number of your data points in each of the bins.
wy is the evaluation of K(x) at the points
wx.*d'. Here d, a p x 1 vector, is the binwidth in each coordinate direction (e.g. as used in the bindata command)

sym specifies the symmetry of the Kernel you use:
You can specify via the parameter sym the symmetry of K(x). This is given by
sym = s_1*20 + s_2*21 + ...+ s_p*2(p-1)
where s_j = 0 indicates that K(x) is an even function with respect to the j-th coordinate, respectively s_j = 1 that it is an odd function.
The default value is sym = (0,...,0).

COMPUTATION:
The convolution is calculated for each bin (b_1...b_j...b_p) of the full grid xc (given by the bins in xb plus the empty bins) and the result is stored in the matrix yc.

The calculation of yc_i involve the values of yb that belong to the bins in a neighbourhood of xc_i, defined by the matrix wx.
The matrix wx is a matrix of (non-negative) integers where wx_l = (k_1...k_j...k_p) indicates that the values of yb belonging to the bins (b_1+/-k_1...b_j+/-k_j... b_p+/-k_p) should be multiplied by the value wy_i (or -wy_i depending on the symmetry considered for K(x)). The sum of all these products give the result yc_i.

The output vector fill is defined according to, fill_i = 1 if xc_i is in xb and fill_i = 0 otherwise.

**Group:** Mathematical Functions
**Subgroup:** Trigonometric Functions
**See also:** acos, acosh, asin, asinh, atan, atan2, atanh, cosh, sin, sinh, tan, tanh

**Function: COS**

**Description:** cos returns the cosine in radian of the elements of an array.

**Usage:** *z = cos (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Example:**
```
x = #(pi/2, pi/4)~#(pi, 3/4*pi)
cos (x)
```

**Result:**
```
[1,] 6.1232e-17    -1
[2,] 0.70711 -0.70711
```

**Group:** Mathematical Functions
**Subgroup:** Trigonometric Functions
**See also:** acos, acosh, asin, asinh, atan, atan2, atanh, cos, sin, sinh, tan, tanh

## Function: cosh

**Description:** cosh returns the cosine hyperbolicus of the elements of an array.

**Usage:** *z = cosh (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Example:**

```
x = #(1, 2)~#(3, 4)
cosh (x)
```

**Result:**

```
[1,] 1.5431 10.068
[2,] 3.7622 27.308
```

**Group:** Plot Manipulation
**See also:** setmaskp, setmaskl, freecolor

## Function: createcolor

**Description:** createcolor allocates the colors for user

**Usage:** *createcolor(rgb)*

**Input:** *rgb* a matrix with 3 columns; the 3 numbers in each row are values from 0 to 256 and content red, green and blue parts of a color. That command prepared the XploRe4 for setmaskl or setmaskp. If you cannot allocate more color, you can help you with freecolor(). The maximal number of different rows in rgb is 248.

**Example:**

```
x=uniform(10000, 2)*2-1
z=x.*x
z=z[,1]+z[,2]
z=z/max(z)
z=z*248
grey=z~z~z
freecolor()
createcolor(grey)
z=x~z
setmaskp(z, grey, 8, 5)
d=createdisplay(1, 1)
show(d, 1, 1, z)
```

**Result:** `you see the points that are nearer to the center darker as the other points`

**Group:** Plot Manipulation
**See also:** show

## Function: **createdisplay**

**Description:** createdisplay create a display for further plotting the datasets or texts.

**Usage:** *di=createdisplay(li_count, col_count)*

**Input:**

| | | |
|---|---|---|
| | *li_count* | integer, requested count of lines of di |
| | *col_count* | integer, requested count of columns of di |

**Output:** *di* Display

**Example:**

```
dd=createdisplay(2,2)
t=(1:60)/10
x=sin(t)
y=sin(2*t)
z=sin(3*t)
show(dd, 1, 1, x~y)
show(dd, 1, 2, x~z)
show(dd, 2, 1, y~z)
show(dd, 2, 2, y~x)
```

**Result:** `you have adisplay with four parts with nice lines in each part.`

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** cumsum, prod, sum

## Function: cumprod

**Description:** cumprod computes the cumulative product of the elements in an array regarding a given dimension.

**Usage:** *z = cumprod (x, j)*

**Input:**

| | |
|---|---|
| *x* | d1 x d2 x ... x dn array |
| *j* | integer (number of dimension, default = 1) |

**Output:**

| | |
|---|---|
| *z* | d1 x d2 x ... x dn array |

**Example:**

```
x = #(1, 2, 3)~#(2, 3, 4)~#(3, 4, 5)
cumprod (x)
```

**Result:**

```
[1,]     1     2     3
[2,]     2     6    12
[3,]     6    24    60
```

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** cumprod, prod, sum

**Function:** **cumsum**

**Description:** cumsum computes the cumulative sum of the elements an array regarding a given dimension.

**Usage:** *z = cumsum (x, j)*

**Input:**

| | | |
|---|---|---|
| | *x* | d1 x d2 x ... x dn array |
| | *j* | integer (number of dimension, default = 1) |

**Output:**

| | | |
|---|---|---|
| | *z* | d1 x d2 x ... x dn array |

**Example:**

```
x=#(1,2,3)~#(3,4,5)~#(5,6,7)
cumsum (x)
```

**Result:**

```
[1,]     1     3     5
[2,]     3     7    11
[3,]     6    12    18
```

**Group:** Time and Date functions
**Subgroup:** Time as string output
**See also:** year month dayofweek hour minute second

## Function: dayofmonth

**Description:** returns the day of the month as string.

**Usage:** *ts = dayofmonth(t)*

**Input:** *t* n x 1 time vector

**Output:** *ts* n x 1 string array

**Example:**

```
yr = 1970 + (1:10)
t  = settime(yr)
ts = dayofmonth(t)
ts
```

**Result:**

```
[ 1,] 01
[ 2,] 01
[ 3,] 01
[ 4,] 01
[ 5,] 01
[ 6,] 01
[ 7,] 01
[ 8,] 01
[ 9,] 01
[10,] 01
```

**Group:** Time and Date functions
**Subgroup:** Time as string output
**See also:** year month dayofmonth hour minute second

## Function: dayofweek

**Description:** returns the day of the week as string.

**Usage:** *ts = dayofweek(t)*

**Input:** *t* n x 1 time vector

**Output:** *ts* n x 1 string array

**Example:**

```
yr = 1970 + (1:10)
t  = settime(yr)
ts = dayofweek(t)
ts
```

**Result:**

```
[ 1,] Fri
[ 2,] Sat
[ 3,] Mon
[ 4,] Tue
[ 5,] Wed
[ 6,] Thu
[ 7,] Sat
[ 8,] Sun
[ 9,] Mon
[10,] Tue
```

**Group:** Flow Control
**See also:** switch case break endsw

## Function: default

**Description:** When the program's counter comes inside a switch construct to the keyword default, the following commands are processed in any case. A default statement can be finished by the keyword break, but does not have to.

**Usage:** *switch case (cond) break default endsw*

**Notes:** Even if the condition is a multidimensional array only the first element is checked for TRUE (!=0) or FALSE (==0)

**Example:**

```
proc (s) = sign (x)
switch
case (x<0)  y = -1  break
case (x>0)  y =  1  break
default     y =  0  break   ; this break has no effect
endsw
endp
```

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** list, append, insert, names

## Function: delete

**Description:** deletes the object at the given position from the list.

**Usage:** *delete (l, pos)*

**Input:**

| | |
|---|---|
| *l* | list |
| *pos* | integer |

**Notes:** Although it is allowed that a list can contain components with the same name, it is not recommended.

**Example:**
```
l = list (1, 2, 3)
delete (l, 1)
l
```

**Result:**
```
Content of object l.el2

[1,]        2
Content of object l.el3

[1,]        3
```

**Group:** Plot Manipulation
**See also:** show, adddata, getdata

## Function: deletedata

**Description:** deletedata removes a dataset from already plotted ones.

**Usage:** *deletedata(di, l_num, c_num, da_num)*

**Input:**

| | |
|---|---|
| *di* | Display |
| *l_num* | integer, 1 ... count of lines of di |
| *c_num* | integer, 1 ... count of columns of di |
| *da_num* | integer, 1 ... count of plotted datasets |

**Example:**

```
di=createdisplay(1, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
show(di, 1, 1, x~y, x~knn(y, 10))
; let have a look of the original data and the smoother
deletedata(di, 1, 1, 2)
; remove the smoother
```

**Result:** `you see only one dataset.`

**Group:** Linear Algebra
**See also:**

**Function:** **det**

**Description:** det computes the determinant of a matrix.

**Usage:** *z = det (x)*

**Input:** *x* n x n matrix

**Output:** *z* scalar

**Notes:** x must be a square matrix, otherwise an error is reported.

**Example:**
```
x=#(1,0)~#(0,1)
det (x)
```

**Result:**
```
[1,]      1
```

**Group:** Matrix Manipulation
**Subgroup:** General Commands
**See also:** matrix, unit

**Function:** **diag**

**Description:** diag creates a diagonal matrix from a vector.

**Usage:** $z = diag\ (x)$

**Input:** $x$ p x 1 matrix

**Output:** $z$ p x p diagonal matrix

**Example:** `diag (1:4)`

**Result:**

```
[1,]     1     0     0     0
[2,]     0     2     0     0
[3,]     0     0     3     0
[4,]     0     0     0     4
```

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** cols, rows

## Function: dim

**Description:** dim gives the vector of the numbers of the elements in each dimension of an array.

**Usage:** $d = dim(x)$

**Input:** $x$ array

**Output:** $d$ vector, containing the numbers of the elements in each dimension

**Notes:** dim(dim(x)) gives the dimension.

**Example:** `dim (matrix(2,3,4))`

**Result:**

```
[1,]      2
[2,]      3
[3,]      4
```

**Group:** Linear Algebra
**See also:** agglom, bindata, grid

## Function: distance

**Description:** distance computes the distance between p-dimensional datapoints depending on a specified metric.

**Usage:** *d = distance (x {, metric})*

**Input:**
*x* n x p matrix
*metric* string indentifying the metric name

**Output:** *d* symmetric n x n matrix of pairwise distances

**Notes:** parameters for metric are: l1, l2, maximum.
If no metric is specified, l2 is used.

**Example:**

```
x = #(1,4)~#(1,5)
distance (x, "l1")
distance (x, "l2")
distance (x, "maximum")
```

**Result:**

```
[1,]     0     7
[2,]     7     0

[1,]     0     5
[2,]     5     0

[1,]     0     4
[2,]     4     0
```

**Group:** Dynamically Linked Library
**See also:** dlopen dlclose dlquery

## Function: dlcall

**Description:** dlcall calls a function defined in a dynamically linked library.

**Usage:** *dlcall ({handle}, n, {x1, x2, ... })*

**Input:**

| | |
|---|---|
| *handle* If it is omitted in all dlls is searched for the function | scalar (optional), handle of the dll to load the function from. |
| *n* | string, name of function |
| *x1* | array (optional), argument for the function |

**Notes:** In contrast to other functions or procedures the objects are passed by reference. That means that the return values of the functions are available in this objects. They must be initialized before the call. It is possible to pass temporary objects (see example).

**Example:**

```
// (if the dll 'randu.so' is loaded)
x = 1:100
dlcall ("randu", x, rows(x), 1)    // the second and third objects are
x
```

**Result:**

```
x contains random numbers generated by the famous RANDU generator.
DO NOT USE THEM AS RANDOM NUMBERS!
It produces already in three dimensions considerably dependencies.
Examine x[1:98]~x[2:99]~x[3:100] in a three-dimensional spin plot.
```

**Group:** Dynamically Linked Library
**See also:** dlopen dlcall dlquery

## Function: dlclose

**Description:** dlclose closes a dynamically linked library loaded by dlopen.

**Usage:** *dlclose* ($\{n\}$)

**Input:**

| | | |
|---|---|---|
| **Input:** | $n$ | scalar, handle of the dll or |
| | $n$ | string, name of the dll |

**Notes:** If no argument is given all opened dlls are closed.

**Example:** `dlclose (2)`

**Result:** `dll with handle 2 is closed (if there is any)`

**Group:** Dynamically Linked Library
**See also:** dlclose dlcall dlquery

## Function: dlopen

**Description:** dlopen loads a dynamically linked library.

**Usage:** *handle = dlopen (n)*

**Input:** *n* string, name of dll

**Output:** *handle* scalar, handle to address this library

**Notes:** The library's handle is used to distinguish between several dlls if they contain functions with the same name.

**Example:**
```
// compile the file randu.c in the examples directory with e.g.
// gcc -shared -o randu.so randu.c
h = dlopen ("randu.so")    ; maybe you must give the full path
```

**Result:** `dll is loaded`

**Group:** Dynamically Linked Library
**See also:** dlopen dlclose dlcall

## Function: dlquery

**Description:** dlquery returns the handles and names of the loaded dlls.

**Usage:** *q = dlquery ()*

**Output:**

| | |
|---|---|
| *q.n* | scalar number of the loaded dlls |
| *q.name* | n vector of strings (optional) containing the names of the dlls |
| *q.handle* | n vector (optional), consists of the handles of the dlls |

**Notes:** if there is no dll loaded, dlquery will only return the number 0, i.e. that q has no component name or handle, respectively.

**Example:**

```
dlquery ()
```

**Result:**

```
0    // or the dll information from the dlls loaded before
```

**Group:** Flow Control
**See also:** endo, until, while

## Function: do

**Description:** do starts a loop until the first element of the argument of until is true. do has to be followed by an until.

**Usage:** *do <xplore commands> until (cond)*

**Example:**

```
proc(j)=factorial(x)
if (x>1)
j=1
do
j=j*x
x=x-1
until (x<2)
else
j=x
endif
endp
```

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** [], list, {}

**Function:** dot

**Description:** a dot gives an element of a list.

**Usage:** $y = x.elem$

| | | |
|---|---|---|
| **Input:** | *x* | list |
| | *elem* | name of an element of x |
| **Output:** | *y* | element of x with name elem |

**Notes:** An Element of a list can be an array or a list.
Another kind to select an element is with the number and {}.

**Example:**
```
x1=#(1,3)~#(3,1)
x2=4:6
y=list(x1,x2)
y.x1
```

**Result:**
```
[1,]     1     3
[2,]     3     1
```

**Group:** Information and Control
**See also:** editmac

## Function: edit

**Description:** edit opens a editor window with the respective matrix.

**Usage:** *edit(x)*

**Input:** *x* matrix

**Example:**
```
x=diag(matrix(4))
edit(x)
```

**Result:** `a window to edit matrix x.`

**Group:** Information and Control
**See also:** edit, savemac

## Function: editmac

**Description:** editmac starts an editor to view or change a macro's definition.

**Usage:** *editmac(name)*

**Input:** *name* string, containing the name of the macro

**Notes:** Only the definition inside XploRe is updated NOT the file, where it may be loaded from.

**Example:**

```
proc(j)=factorial(x)
prod(1:x)
endp

editmac("factorial")
```

**Result:** `a window to edit the macro factorial`

**Group:** Linear Algebra
**See also:** svd

## Function: eigsm

**Description:** eigsm computes the eigenvalues and eigenvectors of a symmetric matrix in unsorted order. As usally this functionality extends to higher dimensional arrays.

**Usage:** *y = eigsm (x)*

**Input:** *x* p x p matrix

**Output:**
*y.values* p x 1 vector with the eigenvalues
*y.vectors* p x p matrix with the normalized eigenvectors

**Notes:** x must be a symmetric matrix. Eigsm doesn't check whether the matrix is symetric or not.

**Example:**

```
x=#(1,3)~#(3,1)
eigsm(x)
```

**Result:**

```
[1,]    -2
[2,]     4

[1,] 0.70711 0.70711
[2,] -0.70711 0.70711
```

**Group:** Flow Control
**See also:** endif, if

## Function: else

**Description:** else branches in if-endif sequences. It is an optional part of this sequences.

**Usage:** *if (cond) <xplore commands> {else <xplore commands>} endif*

**Input:** *cond* n x 1 vector of condition

**Example:**

```
proc(y)=abso(x)
if (x<0)
y=-1.0*x
else
y=x
endif
endp
```

**Group:** Flow Control
**See also:** else, if

**Function:** **endif**

**Description:** endif ends an if-endif sequence. If has to be followed by an endif. Else is an optional part of if sequences.

**Usage:** *if (cond) <xplore commands> {else <xplore commands>} endif*

**Example:**

```
proc(y)=abso(x)
if (x<0)
y=-1.0*x
else
y=x
endif
endp
```

**Group:** Flow Control
**See also:** do, until, while

## Function: endo

**Description:** endo end a while loop. while has to be followed by an endo.

**Usage:** *while (cond) <xplore commands> endo*

**Example:**

```
proc(j)=factorial(x)
if (x>1)
j=1
while (x>=2)
j=j*x
x=x-1
endo
else
j=x
endif
endp
```

**Group:** Flow Control
**See also:** switch case break default

## Function: endsw

**Description:** endsw finishes a switch-endsw block. It serves as branch destination for the different breaks inside the block.

**Usage:** *switch case (cond) break default endsw*

**Notes:** Even if the condition is a multidimensional array only the first element is checked for TRUE (!=0) or FALSE (==0)

**Example:**

```
proc (s) = sign (x)
switch
case (x<0)  y = -1  break
case (x>0)  y =  1  break
default     y =  0  break   ; this break has no effect
endsw
endp
```

**Group:** Statistical Distributions
**See also:** erfc

**Function:** **erf**

**Description:** erf returns the error function.

**Usage:** *y = erf(x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Notes:** erf(x) = 2/sqrt(pi)*integral from 0 to x of exp(-t*t) dt

**Example:**
```
x = #(0, 1/sqrt(2))~#(-1, -1/sqrt(2))
erf (x)
```

**Result:**
```
[1,]       0 -0.8427
[2,] 0.68269 -0.68269
```

**Group:** Statistical Distributions
**See also:** erf

## Function: erfc

**Description:** erfc returns 1.0 - erf(x).

**Usage:** *y = erfc(x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Notes:** erf(x) = 2/sqrt(pi)*integral from 0 to x of exp(-t*t) dt

**Example:**

```
x = #(0, 1/sqrt(2))~#(-1, -1/sqrt(2))
erfc (x)
```

**Result:**

```
[1,]       1 1.8427
[2,] 0.31731 1.6827
```

**Group:** Flow Control
**See also:** break, exit, endp, proc, warning

**Function:** **error**

**Description:** error is used to communicate out of XploRe programs.

**Usage:** *error(x, m)*

**Input:**

| | |
|---|---|
| *x* | condition |
| *m* | string, message |

**Example:**

```
x=matrix(2,2)
error (rows(x) < 5, "rows of x less than 5")
```

**Result:**

```
XploRe_Error_box containing:
CommandLine rows of x less than 5 11Error in Macro
```

**Group:** Information and Control
**See also:** _ execfile

## Function: exec

**Description:** exec executes the given string as it would be typed into the commandline.

**Usage:** *exec(cmd)*

**Input:** *cmd* string, containing the command

**Example:**
```
a="reshape((1:4),#(2,2))"
exec(a)
```

**Result:**
```
[1,]    1    3
[2,]    2    4
```

**Group:** Information and Control
**See also:** _ exec func

## Function: execfile

**Description:** execfile executes all commands contained in the specified file. Files are only seeked in the current directory. If necessary the suffix '.xpl' is appended. If both versions, i.e. with and without suffix, are available the file with suffix will be processed.

**Usage:** *execfile (name)*

**Input:** *name* string containing the file name

**Notes:** if the file could not be found, an error is set. It is allowed to use execfile recursivly but not recommended. The depth of recursion is limited by the number of file descriptor which are provided by the underlying OS.

**Example:** `execfile ("startup")`

**Result:** `returns the startup commands, if the startup file is in the current directory.`

**Group:** Information and Control
**See also:** existglobal

## Function: exist

**Description:** exist try to find if the input object exists or not. If the object exists, exist gives the typ of the object.

**Usage:** *r = exist ("x") or r = exist(x)*

**Input:** *x* object name

**Output:** *r* scalar

**Notes:** The result will be
-1 object doesn't exist
0 object is empty, in macros each not asigned object is empty
1 object is numeric
2 object is text
3 object has type XPLTIME
4 object is a display
9 composed objects (lists)
If the argument of exist denotes a macro, 10 is added. E.G. if there is an object 'cov=1' and the library("xplore") is loaded, exist("cov") results in 11 (10 for proc and 1 for numeric).
exist will not find a global object.
If x is defined and it isn't a macro you can also use exist(x) instead of exist("x"). Then the result can never be -1.

**Example:**
```
x = matrix (2,4)
exist("x")
x = createdisplay(1,1)
exist("x")
exist("y")
```

**Result:**
```
[1,]      1
[1,]      4
[1,]     -1
```

**Group:** Information and Control
**See also:** exist, getglobal, putglobal

## Function: existglobal

**Description:** existglobal checks existince of a global variable

**Usage:** *r = existglobal (name)*

**Input:** *name* name of an arbitrary object

**Output:** *r* scalar

**Notes:** This command is only useful within macros.
The result will be
0 object doesn't exist
1 object is numeric
2 object is text
3 object has type XPLTIME
4 object is a display
9 composed objects (lists)
If the argument of existglobal denotes a macro, 10 is added. E.G. if there is an object 'cov=1' and the library("xplore") is loaded, existglobal("cov") results in 11 (10 for proc and 1 for numeric).

**Example:**
```
e = existglobal ("eulerc")
```

**Result:**
```
e contains either 1 (exists) or 0 (does not exist)
```

**Group:** Mathematical Functions
**Subgroup:** Exponential and Logarithmic Functions
**See also:** eh, exp2, exp10, expm1 log, log2, log10, log1p, ln, lg

## Function: exp

**Description:** exp produces the exponentation to base e=2.7181... of the elements in an array.

**Usage:** *z = exp (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Example:** `exp (0:1)`

**Result:**

```
[1,]      1
[2,] 2.7183
```

**Group:** Mathematical Functions
**Subgroup:** Exponential and Logarithmic Functions
**See also:** exp, log, log10, log1p

## Function: expm1

**Description:** expm1 computes (e**x)-1 accurately even for tiny x.

**Usage:** *z = expm1 (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Example:** `expm1 (0:1)`

**Result:**

```
[1,]      0
[2,] 1.7183
```

**Group:** Mathematical Functions
**Subgroup:** Fourier and Wavelet transforms
**See also:** invfft

## Function: fft

**Description:** fft computes the Fast Fourier Transformation of a complex vector.

**Usage:** *y = fft (x)*

**Input:** *x* p x 2 matrix

**Output:** *y* p x 2 matrix

**Notes:** In the input matrix the first column is the real part of the vector and the second column the imaginary part.
In the output matrix the first column is the real part of the transformed vector and the second column the imaginary part.
p must be an integer power of 2.

**Example:**

```
randomize(0)
x = normal(4)
y = #(0,0,0,0)
z=fft(x~y)
invfft(z)
```

**Result:**

```
[1,] -0.21293     0
[2,] -1.0078 -9.9048e-17
[3,] 1.9502     0
[4,] -1.3052 9.9048e-17
```

**Group:** Mathematical Functions
**Subgroup:** Common Functions
**See also:** ceil

## Function: floor

**Description:** floor gives the next smaller integer value of the elements of an array.

**Usage:** *z = floor (x)*

**Input:** *x* d1 x d2 x .. x dn array

**Output:** *z* d1 x d2 x .. x dn array

**Example:**
```
x = #(0.79, 2.1) ~ #(-3.99, -6.01)
floor (x)
```

**Result:**
```
[1,]      0    -4
[2,]      2    -7
```

**Group:** Information and Control
**See also:** func

## Function: free

**Description:** free removes global objects. It is convinient to delete big objects which consume a lot of memory. If free is invoked without arguments all objects are deleted.

**Usage:** *free (x1, x2, ...)*

**Input:** *x1, ...* objects of arbitrary types

**Notes:** free cannot be used within macro. If you free all objects some libraries will not run any longer.

**Example:**
```
x=matrix(3,3)
free("x")
```

**Result:** `The object x is deleted.`

**Group:** Plot Manipulation
**See also:** createcolor

## Function: freecolor

**Description:** freecolor deallocates the colors that were allocated by user or by system.

**Usage:** *freecolor({option})*

**Input:** *option* a string "user" to free the colors allocated by user with createcolor or "color" to try to free the system color (only by TrueColor-Monitor). By the freecolor() without option the both color classes will be deallocated.

**Example:**

```
x=uniform(10000, 2)*2-1
z=x.*x
z=z[,1]+z[,2]
z=z/max(z)
z=z*248
grey=z~z~z
freecolor()
createcolor(grey)
z=x~z
setmaskp(z, grey, 8, 5)
d=createdisplay(1, 1)
show(d, 1, 1, z)
```

**Result:** `you see the points that are nearer to the center darker as the other points`

**Group:** Reading and Writing
**See also:** execfile, free, library, getenv

## Function: func

**Description:** func loads files and executes them. If necessary the suffix xpl is appended.

**Usage:** *func(name)*

**Input:** *name* string, name of the file

**Notes:** It searches first in the current directory, then in ./lib and then in the directory where the environment 'xpl4lib' points to.

**Example:** `func("examples/switch.xpl")`

**Result:** `read all function defined in XploRe_Home/examples/switch.xpl.`

**Group:** Mathematical Functions
**Subgroup:** Fourier and Wavelet transforms
**See also:** invfwt fwtin fwtinshift invfwtin

**Links:** twstart.html Wavelet tutorial

**Function:** **fwt**

**Description:** fwt computes the Fast Wavelet Transformation of a vector.

**Usage:** *{a, b} = fwt (x, l, h)*

**Input:**

| | |
|---|---|
| *x* | n x 1 vector |
| *l* | scalar |
| *h* | m x 1 vector |

**Output:**

| | |
|---|---|
| *a* | p x 2 matrix |
| *b* | q x 3 matrix |

**Notes:** n has to be a power of 2. l gives the number of coefficients for the father wavelets. n-l is the number of coeffients of the mother wavelets. The matrix a contains in the first column the index of the coefficient of the father wavelet and in the second column the coefficient itself.
The matrix b contains in the first two columns the indices of the coefficient of the mother wavelet and in the third column the coefficient itself.

**Example:**

```
library ("wavelet")
x = (0:1023)/1023
{a, b} = fwt (x, 4, daubechies4)
a
```

**Result:**

```
[1,] 0.000000 0.079014
[2,] 1.000000 0.204137
[3,] 2.000000 0.337344
[4,] 3.000000 0.379505

The matrix b should contain only entries which are approxemately
zero.
```

**Group:** Mathematical Functions
**Subgroup:** Fourier and Wavelet transforms
**See also:** fwt invfwt fwtinshift invfwtin

**Links:** twstart.html Wavelet tutorial

**Function:** **fwtin**

**Description:** fwtin computes the Fast Wavelet Transformation of all circular shifts of the vector x.

**Usage:** *ti = fwtin (x, d, h)*

| | | |
|---|---|---|
| **Input:** | *x* | n x 1 vector |
| | *d* | scalar |
| | *h* | m x 1 vector |
| **Output:** | *ti* | n x d matrix |

**Notes:** n has to be a power of 2. 2d gives the number of coefficients for the father wavelets.

**Example:**

```
; set random seed of random generator
randomize(0)
; load the library wavelet to get the constants
library ("wavelet")
; generate a x
x  = (0:15)/16
; use as y a noisy sine curve
y  = sin(pi*x)+normal(16)
; compute translation invariant coefficients
ti = fwtin (y, 2, daubechies4)
; show them on the screen
ti
```

**Result:**

```
[ 1,] -0.16276   0.1688
[ 2,]  0.13875   2.5233
[ 3,]    1.125  -1.4808
[ 4,] -0.22687   0.4851
[ 5,]  0.94395  0.10797
[ 6,] -0.61274  -1.8444
[ 7,] -0.26754 -0.36924
[ 8,]  0.22756 -0.44845
[ 9,]  -0.4672  0.50097
[10,]   1.4851  -1.8111
[11,] -0.69653 -0.69367
[12,]  0.95768   1.2814
[13,]  0.79618   -1.169
[14,] -0.60832   1.6624
[15,] -0.19415   0.7346
[16,] -0.10738  0.35211
```

**Group:** Mathematical Functions
**Subgroup:** Fourier and Wavelet transforms
**See also:** fwt invfwt fwtin invfwtin

**Links:** twstart.html Wavelet tutorial

## Function: fwtinshift

**Description:** fwtinshift retrieves the averaged coefficient for a fixed number of shifts of the Fast Wavelet Transformation of all circular shifts of the vector x.

**Usage:** *x = fwtinshift (ti, param)*

**Input:**

| | |
|---|---|
| *ti* | n x d matrix |
| *param* | 2 x 1 matrix param[1] = depth (d), param[2] = number of shifts |

**Output:**

| | |
|---|---|
| *x* | n x 1 vector |

**Notes:** n has to be a power of 2. 2d gives the number of coefficients for the father wavelets.

**Example:**

```
library ("wavelet")
x  = (0:15)/16
y  = sin(pi*x)+normal(16)
ti = fwtin (y, 2, daubechies4)
fwtinshift (ti, cols(ti)|0)
```

**Result:** `reproduces x since no shifts done.`

**Group:** Random Variables
**See also:** archest

## Function: genar

**Description:** genar generates an autoregressive process.

**Usage:** *y = genar(x, startval, alpha )*

**Input:**

| | | |
|---|---|---|
| | *x* | n x 1 vector of exogenous variables plus noise |
| | *startval* | p x 1 vector of starting values |
| | *alpha* | p x 1 vector of parameters |

**Output:**

| | | |
|---|---|---|
| | *y* | n x 1 vector |

**Notes:** the process is generated as
y_t = startval, t=1,...,p
y_t = x_t + alpha[1]*y_(t-1) + ... + alpha[p]*y_(t-p), t=p+1,...,n
note that the first p values of x are not used.

**Example:**

```
randomize(0)
x = normal(5)
startval = 0
alpha = 0.1
genar(x, startval, alpha)
```

**Result:**

```
[1,] 0.000000
[2,] -1.007796
[3,] 1.849466
[4,] -1.120232
[5,] -1.546172
```

**Group:** Plot Manipulation
**See also:** show, adddata

## Function: getdata

**Description:** getdata gets a data from a plot.
The data set must be prvious shown in this plot using show oder adddata.

**Usage:** *data=getdata(di, row_num, col_num, data_num)*

**Input:**

| | | |
|---|---|---|
| | *di* | Display |
| | *row_num* | integer, 1, ..., count of lines of di |
| | *col_num* | integer, 1, ..., count of columns of di |
| | *data_num* | integer, 1, ..., count of data sets plotted in this plot |

**Output:**

| | | |
|---|---|---|
| | *data* | a complex object that includes following components: |
| | | a matrix of the points |
| | *data.pm* | a (nx3) matrix with points layout |
| | *data.li* | information about lines, if some points are connected |
| | ... | some more information about labels, selected point, so on. |

**Example:**

```
x=(1:10)~(sin(x/5))
setmaskp(x, 4, 5, 8)
d=createdisplay(1, 1)
show(d, 1, 1, x)
x=1
x=getdata(d, 1, 1, 1)
```

**Result:** `getdata restores the data from the plot and puts it in x.`

**Group:** Information and Control
**See also:** library, setenv

## Function: getenv

**Description:** getenv reads the content of an environment variables (xpl4home, xpl4data, xpl4help, xpl4prog, xpl4backup, format) format contains the output format used by the printf of the C library.

**Usage:** *x = getenv (envname)*

**Input:** *envname* name of the environment "xpl4home" XploRe home directory (is better set by invoking XploRe) "xpl4data" standard directory for loading data "xpl4lib" standard directory for loading librarys "xpl4dll" directory, used by dlopen, to load a dynamically linked library "xpl4help" used by starting the HTML-browser "xpl4prog" standard directory for the program file selection "xpl4backup" directory to back up program files "format" controls the numerical output. Syntax like C's printf "format" controls the numerical output. Syntax like C's printf "stringformat" controls the textual output. Syntax like C's printf "browser" sets the HTML-browser for the help system

**Output:** *x* text containing the value of the envrironment

**Notes:** It is possible to set the enviroment variables for xpl4lib, xpl4dll and xpl4data to more than one directory. In this case, pathnames have to be seperated by :.

**Example:**
```
getenv ("xpl4home")
getenv ("format")
```

**Result:**
```
text of xpl4home is printed to the output window
the output format string (default is "% 8.5lg")
```

**Group:** Information and Control
**See also:** existglobal, putglobal

## Function: getglobal

**Description:** getglobal reads a global variable

**Usage:** *x = getglobal (name)*

**Input:** *name* name of the global object

**Output:** *x* copy of global object ”name”

**Notes:** This command is only useful within macros

**Example:** `x = getglobal ("x")`

**Result:** `x is a copy of the global variable x`

**Group:** Plot Manipulation
**See also:** setgopt

## Function: getgopt

**Description:** getgopt gets the layout of a plot. It is usually used to copy some of all layout components from one plot to another one.

**Usage:** *gopt=getgopt(di, l_num, c_num)*

**Input:**

| | | |
|---|---|---|
| | *di* | Display |
| | *l_num* | integer, 1 ... count of lines of di |
| | *c_num* | integer, 1 ... count of columns of di |

**Output:** *gopt* a composed object containing following components: a component "title" that contents the headline, a component "xlim" that contents limits for x-axis, a component "ylim" that contents limits for y-axis, a component "xoffset" that contents the width of axis border a component "yoffset" that contents the height of axis border a component "xvalue" that contents the values m and k by m+k*x, a component "yvalue" that contents the values m and k by m+k*y, a component "xorigin" that contents origin for tickmark for x-axis, a component "yorigin" that contents origin for tickmark for y-axis, a component "xmajor" that contents major for tickmark for x-axis, a component "ymajor" that contents major for tickmark for y-axis, a component "xlabel" that contents label for x-axis, a component "ylabel" that contents label for y-axis, a component "rotpoint" that contents rotation point , a component "rotcos" that contents rotation cosinus matrix, a component "scal" that contents scale matrix, a component "transl" that contents translation vector

**Example:**

```
di=createdisplay(4, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
show(di, 1, 1, x~y)
show(di, 2, 1, x~y)
show(di, 3, 1, x~y)
show(di, 4, 1, x~y)
;  you see that thanks different width and size of each plot
; the axisborder have different width and height too.
; Futhermore, the height of border is not enough to show
; the axis values and headline.
; We try to improve it:
setgopt(di, 1, 1, "yoffset", 20|20, "xoffset", 5|5)
gopt=getgopt(di, 1, 1)
setgopt(di, 2, 1, gopt)
setgopt(di, 3, 1, gopt)
setgopt(di, 4, 1, gopt)
```

**Result:** `you see it yourself.`

**Group:** Handling of time
**Subgroup:** Generating timepoints
**See also:** settime settimegran settimezero getnow

## Function: getnow

**Description:** getlocalnow returns the actual date and time with correction for the time zone, daylight savings and so on. The corrections are machine dependend.

**Usage:** *t = getlocalnow()*

**Output:** *t* 1 x 1 time vector

**Example:**

```
t = getlocalnow()
dayofmonth(t)+"."+month(t)+"."+year(t)
```

**Result:** `returns the actual date in the (german) format dd.mm.yyyy`

**Group:** Plot Manipulation
**See also:** setmaskp, getdata, show

## Function: getmask

**Description:** gets a mask for points of a dataset.

**Usage:** *mask=getmask(data)*

**Input:** *data* a data matrix

**Output:** *mask* a mask vector that can be bound to another data

**Example:**
```
di=createdisplay(1, 2)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
data=x~y
color=4*matrix(50) | 5*matrix(50)
layout=3*matrix(25) | 4*matrix(25) | 5*matrix(25) | 6*matrix(25)
setmaskp(data, color, layout, 8)
show(di, 1, 1, data)
; you see the first 25 points as red circles,
; second 25 point as red triangles,
; third 25 points as  magenta crosses
; and 25 last points as magenta rhombus.
; now make an other data set
a=(1:100)/6
x=sin(a).*a
y=cos(a).*a
data=x~y
olddata=getdata(di, 1, 1, 1)
mask=getmask(olddata)
;  now copy the layout of the old data
; to the layout of the neu data
setmaskp(data, mask)
show(di, 1, 2, data)
```

**Result:** `The layouts from the both datasets are same.`

**Group:** Handling of time
**Subgroup:** Generating timepoints
**See also:** settime settimegran settimezero getlocalnow

## Function: getnow

**Description:** getnow returns the actual date and time in Greenwich time.

**Usage:** $t$ = *getnow()*

**Output:** $t$ 1 x 1 time vector

**Example:**
```
t = getnow()
dayofmonth(t)+"."+month(t)+"."+year(t)
```

**Result:** `returns the actual date in the (german) format dd.mm.yyyy`

**Group:** Binning, grids and sequences
**See also:** bindata, aseq, conv, distance, mseq

## Function: grid

**Description:** This command generates a grid with origin x and stepwidth h with respect to each dimension, n indicating the number of gridpoints in the respective dimension.

**Usage:** *z = grid (x, h, n)*

**Input:**
*x* m x 1 x d1 x ... x dn array
*h* m x 1 x d1 x ... x dn array
*n* m x 1 x d1 x ... x dn array

**Output:** *z* m x (n1*n2*...*nm) x d1 x ... x dn array

**Notes:** The coordinates of n all have to be greater than 0.

**Example:**
```
x = # (0,   0,   0)
h = # (1, 2, 3)
n = # (2,   2,   2)
z = grid(x,h,n)
sort(z)
```

**Result:**
```
0    2     3
0    0     3
0    0     0
0    2     0
1    2     0
1    0     3
1    0     0
1    2     3
```

**Example:**
```
x = # (0,   0,   0)
h = # (0.5, 0.5, 0.5)
n = # (1,   2,   3)
z = grid(x,h,n)
sort(z)
```

**Result:**
```
0    0      0
0    0.5    0
0    0      0.5
0    0.5    0.5
0    0      1
0    0.5    1     .
```

**Group:** Time and Date functions
**Subgroup:** Time as string output
**See also:** year month dayofmonth dayofweek minute second

## Function: nhour

**Description:** returns the hour as a string

**Usage:** *ts = nhour(t)*

**Input:** *t* n x 1 time vector

**Output:** *ts* n x 1 string array

**Example:**

```
yr = 1970 + (1:10)
t  = settime(yr)
ts = hour(t)
ts
```

**Result:**

```
[ 1,] 00
[ 2,] 00
[ 3,] 00
[ 4,] 00
[ 5,] 00
[ 6,] 00
[ 7,] 00
[ 8,] 00
[ 9,] 00
[10,] 00
```

**Group:** Flow Control
**See also:** else, endif

**Function: if**

**Description:** if does conditional branching, if the first element of the argument is true. If has to be followed by an endif. Else is an optional part of if sequences.

**Usage:** *if (cond) <xplore commands> {else <xplore commands>} endif*

**Input:** *cond* n-dimensional array of condition

**Notes:** Even if the condition is a multidimensional array only the first element is checked for TRUE (!=0) or FALSE (==0)

**Example:**

```
proc(y)=abso(x)
if (x<0)
y=-1.0*x
else
y=x
endif
endp
```

**Group:** Matrix Manipulation
**Subgroup:** General Commands
**See also:** mix, paf, sort

## Function: index

**Description:** index generates a new matrix z from an old matrix x by extracting the rows of x indicated in the index vector i.

**Usage:** *y = index (x, ind)*

**Input:**

| | |
|---|---|
| *x* | n x p matrix |
| *ind* | k dimensional vector |

**Output:**

| | |
|---|---|
| *y* | k x p matrix |

**Notes:** For elements k of the index vector that verify k $<$ 1 or k $>$ n ,a nan is returned.

**Example:**

```
x = #(0.1, 0.2, 0.3, 0.4)
i = #(4, 3, 2, 2, 1)
( index (x, i) )'
```

**Result:**

```
[1,]   0.4   0.3   0.2   0.2   0.1
```

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** list, append, delete, names

**Function:** **insert**

**Description:** inserts an object to the specified position within a list.

**Usage:** *insert (l, pos, x)*

| **Input:** | | |
|---|---|---|
| | *l* | list |
| | *pos* | integer |
| | *x* | object |

**Notes:** Although it is allowed that a list can contain components with the same name, it is not recommended.

**Example:**
```
l = list (1, 2)
insert (l, 2, 44)
l
```

**Result:**
```
Content of object l.el1

[1,]        1
Content of object l.el2

[1,]       44
Content of object l.el2

[1,]        2
```

**Group:** Linear Algebra
**See also:** ', trans

## Function: inv

**Description:** inv computes the inversematrix. As usally this functionality extends to higher dimensional arrays.

**Usage:** *z = inv (x)*

**Input:** *x* array of square matrices

**Output:** *z* array of square matrices

**Notes:** x must be an array of nonsingular square matrices.

**Example:**

```
x = #(1,2)~#(3,4)
inv (x)
```

**Result:**

```
[1,]    -2   1.5
[2,]     1  -0.5
```

**Group:** Mathematical Functions
**Subgroup:** Fourier and Wavelet transforms
**See also:** fft

## Function: invfft

**Description:** invfft computes the Inverse Fast Fourier Transformation of a complex vector.

**Usage:** *y = invfft (x)*

**Input:** *x* p x 2 matrix

**Output:** *y* p x 2 matrix

**Notes:** In the input matrix the first column is the real part of the vector and the second column the imaginary part.
In the output matrix the first column is the real part of the transformed vector and the second column the imaginary part.
p must be an integer power of 2.

**Example:**

```
randomize(0)
x = normal(4)
y = #(0,0,0,0)
z=fft(x~y)
invfft(z)
```

**Result:**

```
[1,] -0.21293     0
[2,] -1.0078 -9.9048e-17
[3,] 1.9502     0
[4,] -1.3052 9.9048e-17
```

**Group:** Mathematical Functions
**Subgroup:** Fourier and Wavelet transforms
**See also:** invfwt fwtin fwtinshift invfwtin

**Links:** twstart.html Wavelet tutorial

**Function:** **fwt**

**Description:** fwt computes the Fast Wavelet Transformation of a vector.

**Usage:** *{a, b} = fwt (x, l, h)*

**Input:**

| | |
|---|---|
| *x* | n x 1 vector |
| *l* | scalar |
| *h* | m x 1 vector |

**Output:**

| | |
|---|---|
| *a* | p x 2 matrix |
| *b* | q x 3 matrix |

**Notes:** n has to be a power of 2. l gives the number of coefficients for the father wavelets. n-l is the number of coeffients of the mother wavelets. The matrix a contains in the first column the index of the coefficient of the father wavelet and in the second column the coefficient itself.
The matrix b contains in the first two columns the indices of the coefficient of the mother wavelet and in the third column the coefficient itself.

**Example:**

```
library ("wavelet")
x = (0:1023)/1023
{a, b} = fwt (x, 4, daubechies4)
a
```

**Result:**

```
[1,] 0.000000 0.079014
[2,] 1.000000 0.204137
[3,] 2.000000 0.337344
[4,] 3.000000 0.379505

The matrix b should contain only entries which are approxemately
zero.
```

**Group:** Mathematical Functions
**Subgroup:** Fourier and Wavelet transforms
**See also:** fwt invfwt fwtin fwtinshift

**Links:** twstart.html Wavelet tutorial

**Function:** **invfwtin**

**Description:** fwtin computes the average of the inverse Fast Wavelet Transformation of all circular shifts from ti.

**Usage:** *x = invfwtin (ti, d, h)*

| | | |
|---|---|---|
| **Input:** | *ti* | n x d matrix |
| | *d* | scalar |
| | *h* | m x 1 vector |
| **Output:** | *x* | n x 1 vector |

**Notes:** n has to be a power of 2. 2d gives the number of coefficients for the father wavelets.

**Example:**

```
; set random seed of random generator
randomize(0)
; load the library wavelet to get the constants
library ("wavelet")
; generate a x
x  = (0:15)/16
; use as y a noisy sine curve
y  = sin(pi*x)+normal(16)
; compute translation invariant coefficients
ti = fwtin (y, 2, daubechies4)
; make a small hardthresholding
ti = ti.*(abs(ti).>0.5)
; transform back to estimated data
yh = invfwtin (ti, 2, daubechies4)
; compare original and thresholded data
y~yh
```

**Result:**

```
[ 1,] -0.21293 -0.081272
[ 2,] -0.81271 -0.63678
[ 3,]   2.3329   2.2874
[ 4,] -0.74961 -0.81817
[ 5,] -0.72704  -0.7112
[ 6,]   1.5296   1.5929
[ 7,]  0.53442  0.39329
[ 8,] -0.59385 -0.36972
[ 9,]  0.73405  0.71432
[10,]   1.1803   1.1917
[11,]  -1.5795  -1.6033
[12,]  0.33883  0.36711
[13,] -0.51739  -0.2067
[14,]  0.13434  0.0072323
[15,] -0.34705 -0.056144
[16,]  0.40373  0.32672
```

**Group:** Statistical Data Analysis
**Subgroup:** Nonparametric Methods
**See also:** sker sknn l1line lowess

## Function: isoreg

**Description:** isoreg computes the isotonic regression smoother via the Pool Adjacent Violators algorithm. Given a data set {(X_i,Y_i)} where X_i <= X_(i+1) i=1,...,n finds the values {mhat(X_i)} i=1,...,n, such that, minimizes (1/n) sum_i=1,n [Y_i) - mhat(X_i)]2 subject to mhat(X_i) <= mhat[X_(i+1)], i=1,...,n

**Usage:** *yh = isoreg (y)*

**Input:** *y* array

**Output:** *yh* array

**Notes:** the array y should be sorted according to x.

**Example:**

```
x = read ("geyser")
x = sort (x, #(1,2))
mh = x[,1]~isoreg (x[,2])
d = createdisplay (1,1)
show (d, 1, 1, x, mh)
```

**Result:** `shows the geyser data and the isotonic regression smoother`

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** mknn

## Function: knn

**Description:** computes a running mean over (2k+1) consecutive values of a given vector. To have the same length at the beginning and at the end the first and last value are repeated k times.

**Usage:** $m = knn\ (x,\ k)$

**Input:**

| | |
|---|---|
| $x$ | n vector of the obervations |
| $k$ | scalar range of the running mean |

**Output:**

| | |
|---|---|
| $m$ | n vector containing the smoothed values |

**Example:**

```
randomize(1)
knn (uniform (8), 2)
```

**Result:**

```
[1,]  0.25989
[2,]  0.25989
[3,]  0.25989
[4,]  0.28321
[5,]  0.28438
[6,]  0.31511
[7,]  0.31511
[8,]  0.31511
```

**Group:** Statistical Data Analysis
**Subgroup:** Nonparametric Methods
**See also:** sker isoreg rmed lowess

## Function: l1line

**Description:** l1line computes the least absolute deviation line from scatterplot data. It gives the estimate b0 and b1 that minimizes sum_i=1,n |y_i - b0 - b1 x_i |.

**Usage:** *b = l1line (x, y)*

**Input:**

| | | |
|---|---|---|
| | *x* | n x 1 vector |
| | *y* | n x 1 vector |

**Output:**

| | | |
|---|---|---|
| | *b* | 2 x 1 vector |

**Notes:** If you have ties then XploRe may hang up.

**Example:**

```
; read the geyser data
x = read ("geyser")
; compute the estimate
b = l1line (x)
```

**Result:** `gives coefficient b0=34.73 and b1=10.42.`

**Example:**

```
; compute the least square line
yh = (matrix(rows(x))~x[,1])*b
; create a display for plotting
d = createdisplay(1,1)
; show the data and the fitted line
show (d, 1, 1, x, x[,1]~yh)
```

**Group:** Mathematical Functions
**Subgroup:** Exponential and Logarithmic Functions
**See also:**

## Function: lgamma

**Description:** lgamma computes logarithm of the gamma function.

**Usage:** *l = gamma(x)*

**Input:** *x* n x 1 vector, x > 0 elementwise

**Output:** *l* n x 1 vector, natural logarithm of gamma(x)

**Example:**
```
lgamma(5)
```

**Result:**
```
[1,]   3.1781
```

**Group:** Reading and Writing
**See also:** func, getenv, setenv

## Function: library

**Description:** library loads an xplore library

**Usage:** *library(x)*

**Input:** *x* string, name of library

**Notes:** If the extension ".lib" is not given it is automatically appended.
The library is searched in the directories contained in the environment xpl4lib and in the working directory.

**Example:** `library("xplore")`

**Result:** `loads the xplore macros`

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** comp, dot, size, {}, append, insert, delete, names

## Function: list

**Description:** list generates lists from given objects. If an object is temporary the name of the component is el<position>, otherwise the name of the object at this position.

**Usage:** *l = list(x1 {,x2{,...}})*

**Input:** *xi* element of the list

**Output:** *y* list

**Notes:** Although it is allowed that a list can contain components with the same name, it is not recommended.

**Example:**

```
x = 1
l = list (x, 4)
l
```

**Result:**

```
Content of object l.x

[1,]        1
Content of object l.el2

[1,]        4
```

**Group:** Data Analysis Methods
**See also:** sker, rmed, lowess, locpol

## Function: loch

**Description:** loch computes the local polynomial estimator without mixed terms but allows for including a linear part in the regression model. It is using the quartic kernel.

**Usage:** *loch(x, xd, xg, y, h, logi)*

**Input:**

| | | |
|---|---|---|
| | *x* | n x d matrix nonlinear inputs |
| | *xd* | n x dis matrix inputs for the linear part |
| | *xg* | m x d matrix grid points at which the estimation will be done |
| | *y* | n x q matrix dependent variable |
| | *h* | d x 1 matrix vector of bandwidthes |
| | *logi* | logical scalar 0 if linear, 1 if quadratic polynomials are wished |

**Output:** *mh* m x dd x q matrix regression of the nonlinear part and its v'th derivatives divided by v at xg in the following order: regresion function, first derivatives respectively x1, x2, x3, ..., second derivative*0.5 respectively x1, x2, x3, ...

**Notes:** The data have to be sorted by x and xg !

**Example:**

```
logi=1            ; we aim to use local quadratic
d   =2            ; dimension of nonlinear inputs
dis =1            ; dimension of discrete influence
q   =1            ; dimension of response
n   = 100         ; number of observations
;*****  create data *****
x  = uniform(n,d)
xd = uniform(n,dis)
xd =  (xh.> (1/3))
y  =  1.5*x[,1]^2 -x[,2]^3 +xd*(0.5) +normal(n,1)*0.2
h  = 0.4*matrix(d,1)         ; bandwidth vector
xs = sort(x~xd~y,1)
x  = xs[,1:d]
xd = xs[,(d+1):(d+dis)]
y  = xs[,d+dis+1]
xg = x                       ; 'grid'
mh = loch(x,xd,xg,h,logi)
mh[,1]                       ; regression function
mh[,2]                       ; derivative resp. to x_1
mh[,3]                       ; derivative resp. to x_2
mh[,4]                       ; derivative*0.5 resp. to x_1 x_1
mh[,5]                       ; ...                     x_2 x_2
```

**Group:** Data Analysis Methods
**See also:** sker, rmed, lowess, loch

## Function: locpol

**Description:** locpol computes the local polynomial estimator. It is using the quartic kernel.

**Usage:** *locpol(x, xg, y, h, logi)*

**Input:**

| | |
|---|---|
| *x* | n x d matrix explanatory variables |
| *xg* | m x d matrix grid points at which the estimation will be done |
| *y* | n x q matrix dependent variable |
| *h* | d x 1 matrix vector of bandwidthes |
| *logi* | logical scalar 0 if linear, 1 if quadratic polynomials are wished |

**Output:**

| | |
|---|---|
| *mh* | m x dd x q matrix regression function and its v'th derivatives divided by v at xg in the following order: regression function, first derivatives respectively x1, x2, x3, ... , derivatives respectively x1x1, x1x2, ..., x2x2, x2x3, ... |

**Notes:** The data have to be sorted by x and xg !

**Example:**

```
logi=1            ; we aim to use local quadratic
d   =2            ; dimension of nonlinear inputs
q   =1            ; dimension of response
n   = 100         ; number of observations
;*****  create data *****
x  = uniform(n,d)
y  =  1.5*x[,1]^2 -x[,2]^3 +normal(n,1)*0.2
h  = 0.4*matrix(d,1)          ; bandwidth vector
xs = sort(x~y,1)
x  = xs[,1:d]
y  = xs[,d+1]
xg = x                        ; 'grid'
mh = locpol(x,xg,h,logi)
mh[,1]                        ; regression function
mh[,2]                        ; derivative resp. to x_1
mh[,3]                        ; derivative resp. to x_2
mh[,4]                        ; derivative*0.5 resp. to x_1 x_1
mh[,5]                        ; ...                     x_1 x_2
mh[,6]                        ; ...                     x_2 x_2
```

**Group:** Mathematical Functions
**Subgroup:** Exponential and Logarithmic Functions
**See also:** exp, exp2, exp10, expm1, log2, log10, log1p, ln, lg

## Function: log

**Description:** log returns the natural logarithm of the elements of an array.

**Usage:** $z = log\ (x)$

**Input:** $x$ d1 x d2 x ... x dn array

**Output:** $z$ d1 x d2 x ... x dn array

**Notes:** if this function is applied to negative values it will return -Inf.

**Example:** `log (exp((1:5)'))`

**Result:**

```
[1,]     1     2     3     4     5
```

**Group:** Mathematical Functions
**Subgroup:** Exponential and Logarithmic Functions
**See also:** exp, exp2, exp10, expm1, lg, ln, log, log1p, log2

## Function: log10

**Description:** log10 returns the logarithm base 10 of the elements of an array.

**Usage:** *z = log10 (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Notes:** if this function is applied to negative values it will return -Inf.

**Example:** `log10 (#(10, 100)')`

**Result:**
```
[1,]      1      2
```

**Group:** Mathematical Functions
**Subgroup:** Exponential and Logarithmic Functions
**See also:** exp, exp2, exp10, expm1, lg, ln, log, log2, log10

## Function: log1p

**Description:** log1p computes log(1+x) accurately even for tiny x.

**Usage:** *z = log1p (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Notes:** if this function is applied to negative values it will return -Inf.

**Example:** `log1p (expm1((1:5)'))`

**Result:**

```
[1,]     1     2     3     4     5
```

**Group:** Statistical Data Analysis
**Subgroup:** Nonparametric Methods
**See also:** sker isoreg rmed l1line

## Function: lowess

**Description:** lowess computes the robust locally weighted regression. Fitted values are computed at each of the given values x.

**Usage:** *yh = lowess (x, y{, f})*

**Input:**

| | |
|---|---|
| *x* | n x 1 vector |
| *y* | n x 1 vector |
| *f* | scalar |

**Output:**

| | |
|---|---|
| *yh* | n x 1 vector |

**Notes:** The data must be sorted by x. The span f has to verify 0.2 <= f <= 0.8 (default f=0.5).

**Example:**

```
; read the geyser data
x = read ("geyser")
; sort them
x = sort(x)
; compute the estimate
yh = lowess (x[,1], x[,2])
; create a display for plotting
d = createdisplay(1,1)
; show the data and the fitted line
show (d, 1, 1, x, x[,1]~yh)
```

**Group:** Linear Algebra
**See also:**

## Function: ludecomp

**Description:** ludecomp computes the lu decomposition of a matrix.

**Usage:** *z = ludecomp (x)*

**Input:**

| | |
|---|---|
| *x* | square matrix |

**Output:**

| | |
|---|---|
| *z.l* | containing the lower matrix |
| *z.u* | containing the upper matrix |
| *z.index* | containing an indexvector |

**Notes:** x must be a square matrix.
z.index is needed to rebuild the orign matrix x from l and u.
To rebuild x use index(z.l*z.u, z.index).

**Example:**

```
x = #(1, 2, 3)~#(2, 3, 4)~#(3, 4, 5)
z = ludecomp (x)
z
index(z.l*z.u, z.index)
```

**Result:**

```
[1,]       1       0       0
[2,] 0.33333       1       0
[3,] 0.66667     0.5       1

[1,]       3       4       5
[2,]       0 0.66667  1.3333
[3,]       0       0 1.1102e-16

[1,]       2
[2,]       3
[3,]       1

[1,]       1       2       3
[2,]       2       3       4
[3,]       3       4       5
```

**Group:** Reading and Writing
**See also:**

## Function: makehtml

**Description:** makehtml invokes the automatic translator for generating HTML pages.

**Usage:** *makehtml(n)*

**Input:** *n* name of the file that should be used

**Notes:** The function expects a file name where the procedure is defined. The HTML file is stored in the directory <XPL4HOME>/help if it exists. The command is equivalent to the 'Make Help' button of the editor window.

**Group:** Matrix Manipulation
**Subgroup:** General Commands
**See also:** diag, paf, unit, #

**Function:** **matrix**

**Description:** matrix generates arrays up to eight dimensions.

**Usage:** *z = matrix (d1 {, ..., dn})*

**Input:** *d1, ..., dn* scalar (number of elements in dimension 1 ... n)

**Output:** *z* d1 x d2 x ... x dn matrix containing 1

**Example:** `matrix(2,5)`

**Result:**

```
[1,]     1     1     1     1     1
[2,]     1     1     1     1     1
```

**Group:** Statistical Data Analysis
**Subgroup:** Basic Statistics
**See also:** maxind, median, min, minind, quantile

## Function: max

**Description:** max computes the maximal value of the elements of an array regarding a given dimension.

**Usage:** *z = max (x {,j})*

| | | |
|---|---|---|
| **Input:** | *x* | d1 x d2 x ... x dn array |
| | *j* | integer (number of dimension, default = 1) |
| **Output:** | *z* | d1 x d2 x .. x d(j-1) x 1 x d(j+1) x .. x dn array |

**Example:**

```
randomize (0)
x = normal (2, 4)
max (x,2)
```

**Result:**

```
[1,] 1.9502
[2,] 0.69812
```

**Group:** Statistical Data Analysis
**Subgroup:** Basic Statistics
**See also:** max, median, min, minind, quantile

**Function:** **maxind**

**Description:** maxind gives a row vector with the index of the maximum element in each column of a matrix. As usally this functionality extends to higher dimensional arrays.

**Usage:** $z = maxind\ (x)$

**Input:** $x$ n x p matrix

**Output:** $z$ p dimensional vector

**Example:** `maxind(#(1,3)~#(4,-1))`

**Result:** `[1,]      2      1`

**Group:** Statistical Data Analysis
**Subgroup:** Basic Statistics
**See also:** median, ssd, sum, var

## Function: mean

**Description:** mean computes the mean of the elements of an array regarding a given dimension.

**Usage:** *z = mean (x {,j})*

| | | |
|---|---|---|
| **Input:** | *x* | d1 x d2 x .. x dn array |
| | *j* | integer (number of dimension, default = 1) |
| **Output:** | *z* | d1 x d2 x .. x d(j-1) x 1 x d(j+1) x .. x dn array |

**Example:**

```
randomize (0)
x = uniform (2, 4)
mean (x,2)
```

**Result:**

```
[1,] 0.13953
[2,] 0.31505
```

**Group:** Statistical Data Analysis
**Subgroup:** Basic Statistics
**See also:** max, maxind, median, minind, quantile

## Function: min

**Description:** min computes the minimum value of the elements of an array regarding a given dimension.

**Usage:** *z = min (x {,d})*

**Input:**

| | | |
|---|---|---|
| **Input:** | *x* | d1 x d2 x .. x dn array |
| | *j* | integer (number of dimension, default = 1) |

**Output:**

| | | |
|---|---|---|
| **Output:** | *z* | d1 x d2 x .. x d(j-1) x 1 x d(j+1) x .. x dn array |

**Example:**

```
randomize (0)
x = normal (2, 4)
min (x,2)
```

**Result:**

```
[1,] -1.4341
[2,] -1.5746
```

**Group:** Statistical Data Analysis
**Subgroup:** Basic Statistics
**See also:** max, maxind, median, min, quantile

## Function: minind

**Description:** minind gives a row vector with the index of the minimum element in each column of a matrix. As usally this functionality extends to higher dimensional arrays.

**Usage:** *z = minind (x)*

**Input:** *x* n x p matrix

**Output:** *z* p dimensional vector

**Example:** `minind(#(1,3)~#(4,-1))`

**Result:**
```
[1,]     1     2
```

**Group:** Time and Date functions
**Subgroup:** Time as string output
**See also:** year month dayofmonth dayofweek hour second

## Function: nminute

**Description:** returns the minute as a string

**Usage:** *ts = minute(t)*

**Input:** *t* n x 1 time vector

**Output:** *ts* n x 1 string array

**Example:**
```
t  = settime()
ts = minute(t)
```

**Example:**
```
yr = 1970 + (1:10)
t  = settime(yr)
ts = minute(t)
ts
```

**Result:**
```
[ 1,] 00
[ 2,] 00
[ 3,] 00
[ 4,] 00
[ 5,] 00
[ 6,] 00
[ 7,] 00
[ 8,] 00
[ 9,] 00
[10,] 00
```

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** knn

## Function: mknn

**Description:** mknn gives an array containing the indices of the k nearest neighbors of each element in matrix y with respect to x. The parameter y is optional, if it is not given, x will be taken for y. As a measure of distance the squared euclidean distance is used.

**Usage:** *z = mknn (x, k{, y})*

| | | |
|---|---|---|
| **Input:** | *y* | m x p x d1 x ... x dn array |
| | *x* | n x p x d1 x ... x dn array |
| | *k* | scalar (number of neighbors) |
| **Output:** | *z* | m x k x d1 x ... x dn array |

**Example:**
```
x=1:4
y=2:5
mknn(x,3,y)
```

**Result:**
```
[1,]      2     1     3
[2,]      3     2     4
[3,]      4     3     2
[4,]      4     3     2
```

**Group:** Time and Date functions
**Subgroup:** Time as string output
**See also:** year dayofmonth dayofweek hour minute second

## Function: month

**Description:** returns the month as a string

**Usage:** *ts = month(t)*

**Input:** *t* n x 1 time vector

**Output:** *ts* n x 1 string array

**Example:**

```
yr = 1970 + (1:10)
t  = settime(yr)
ts = month(t)
ts
```

**Result:**

```
[ 1,] Jan
[ 2,] Jan
[ 3,] Jan
[ 4,] Jan
[ 5,] Jan
[ 6,] Jan
[ 7,] Jan
[ 8,] Jan
[ 9,] Jan
[10,] Jan
```

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** list, append, delete, insert

## Function: names

**Description:** gives the names of all components of a list object.

**Usage:** *names (l)*

**Input:** *l* list

**Example:**
```
l = list (11, 99)
names (l)
```

**Result:**
```
[1,] el1
[2,] el2
```

**Group:** Time and Date functions
**Subgroup:** Time as numerical output
**See also:** nyear nmonth ndayofweek nhour nminute nsecond

## Function: ndayofmonth

**Description:** returns the day of the month as a number (1-31)

**Usage:** *tn = ndayofmonth(t)*

**Input:** *t* n x 1 time vector

**Output:** *tn* n x 1 array

**Example:**

```
yr = 1970 + (1:10)
t  = settime(yr)
ts = ndayofmonth(t)
ts
```

**Result:**

```
[ 1,] 1.000000
[ 2,] 1.000000
[ 3,] 1.000000
[ 4,] 1.000000
[ 5,] 1.000000
[ 6,] 1.000000
[ 7,] 1.000000
[ 8,] 1.000000
[ 9,] 1.000000
[10,] 1.000000
```

**Group:** Time and Date functions
**Subgroup:** Time as numerical output
**See also:** nyear nmonth ndayofmonth nhour nminute nsecond

## Function: ndayofweek

**Description:** returns the day of the week as a number (1-7, sunday-saturday)

**Usage:** *tn = ndayofweek(t)*

**Input:** *t* n x 1 time vector

**Output:** *tn* n x 1 array

**Example:**

```
yr = 1970 + (1:10)
t  = settime(yr)
ts = ndayofweek(t)
ts
```

**Result:**

```
[ 1,] 6.000000
[ 2,] 7.000000
[ 3,] 2.000000
[ 4,] 3.000000
[ 5,] 4.000000
[ 6,] 5.000000
[ 7,] 7.000000
[ 8,] 1.000000
[ 9,] 2.000000
[10,] 3.000000
```

**Group:** Optimizer
**See also:** bfgs

## Function: nelmin

**Description:** nelmin searchs a minimum of a function. In each iteration step the function is evaluated at a simplex consisting of p+1 points. The simplex contracts until the variance of the evaluated function values is less than eps (or the maximal number of iterations is reached).

**Usage:** *x = nelmin (x0, f, maxiter {,eps , step})*

**Input:**

| | |
|---|---|
| *x0* | p x n matrix with n starting vectors |
| *f* | text, name of the procedure where the function is defined |
| *maxiter* | integer, maximal number of iterations |
| *eps* | scalar |
| *step* | scalar, lenght of initial simplex |

**Output:**

| | |
|---|---|
| *x.minimum* | p x n matrix with the n minima |
| *x.iter* | number of iterations |
| *x.converged* | 1 if the algorithm has converged with every starting vector and 0 if it has not |

**Notes:** The algorithm stops if the variance of the functionvalues of the points of the simplex is less then eps or the number of iterations is greater then maxiter.
maxiter should be greater then the number of dimensions p.

**Example:**

```
proc (y) = f (x)
y = sum(x^2)
endp
x0    = #(1,1,1)~#(1,2,3)
nelmin (x0, "f", 100, 1.0e-6)
```

**Result:**

```
Content of object nelmin.minimum

[1,] 0.017404 -0.02019
[2,] -0.0070216 0.022459
[3,] 0.016622 0.0042336
Content of object nelmin.iter

[1,]    53
Content of object nelmin.converged

[1,]     1
```

**References:** Nelder, J. A. and Mead, R. (1965). A Simplex Method for Function Minimization. "Computer J.", vol 7,308-313.

**Group:** Time and Date functions
**Subgroup:** Time as numerical output
**See also:** nyear nmonth ndayofmonth ndayofweek nminute nsecond

## Function: nhour

**Description:** returns the hour as a number (0-23)

**Usage:** *tn = nhour(t)*

**Input:** *t* n x 1 time vector

**Output:** *tn* n x 1 array

**Example:**

```
yr = 1970 + (1:10)
t  = settime(yr)
ts = nhour(t)
ts
```

**Result:**

```
[ 1,] 0.000000
[ 2,] 0.000000
[ 3,] 0.000000
[ 4,] 0.000000
[ 5,] 0.000000
[ 6,] 0.000000
[ 7,] 0.000000
[ 8,] 0.000000
[ 9,] 0.000000
[10,] 0.000000
```

**Group:** Time and Date functions
**Subgroup:** Time as numerical output
**See also:** nyear nmonth ndayofmonth ndayofweek nhour nsecond

## Function: nminute

**Description:** returns the minute as a number (0-59)

**Usage:** *tn = nminute(t)*

**Input:** *t* n x 1 time vector

**Output:** *tn* n x 1 array

**Example:**
```
yr = 1970 + (1:10)
t  = settime(yr)
ts = nminute(t)
ts
```

**Result:**
```
[ 1,] 0.000000
[ 2,] 0.000000
[ 3,] 0.000000
[ 4,] 0.000000
[ 5,] 0.000000
[ 6,] 0.000000
[ 7,] 0.000000
[ 8,] 0.000000
[ 9,] 0.000000
[10,] 0.000000
```

**Group:** Time and Date functions
**Subgroup:** Time as numerical output
**See also:** nyear ndayofmonth ndayofweek nhour nminute nsecond

## Function: nmonth

**Description:** returns the month as a number

**Usage:** *tn = nmonth(t)*

**Input:** *t* n x 1 time vector

**Output:** *tn* n x 1 array

**Example:**

```
yr = 1970 + (1:10)
t  = settime(yr)
ts = nmonth(t)
ts
```

**Result:**

```
[ 1,] 1.000000
[ 2,] 1.000000
[ 3,] 1.000000
[ 4,] 1.000000
[ 5,] 1.000000
[ 6,] 1.000000
[ 7,] 1.000000
[ 8,] 1.000000
[ 9,] 1.000000
[10,] 1.000000
```

**Group:**
**See also:** nninit nnvisu

**Function:** **nnfunc**

**Description:** nnfunc computes for a given feed forward network the result for a datavector x.

**Usage:** *y = nnfunc (wei, unit, x)*

**Input:**

| | | |
|---|---|---|
| | *wei* | n x 3 matrix of weights |
| | *unit* | m x 2 matrix of units |
| | *x* | p x k input values |

**Output:**

| | | |
|---|---|---|
| | *y* | p x l output values |

**Example:**

```
x = read ("kredit")        ; read kredit data
y = x[1:10,21]
x = x[1:20,1:20]
;; build logit model
wei  = (1:20)~(21.*matrix(20))~normal(20)
unit = trans(#(-1,0)).*matrix(21)|trans(#(1,2))
ind  = nninit (wei, unit)
wei  = index (wei, ind)
randomize(0)
yh   = nnfunc (wei, unit, x)
y~yh
```

**Result:** `Content of object tmp`

**Group:**
**See also:** nnfunc nnvisu

## Function: nninit

**Description:** nninit checks if a given network is feedforward network and suggest reorderings.

**Usage:** *ind = nninit (wei, unit)*

**Input:**

| | |
|---|---|
| *wei* | n x 3 matrix of weights |
| *unit* | m x 2 matrix of units |

**Output:**

| | |
|---|---|
| *ind* | n x 1 reordering vector |

**Example:**

```
x = read ("kredit")       ; read kredit data
y = x[,21]
x = x[,1:20]
;; build logit model
wei  = (1:20)~(21.*matrix(20))~normal(20)
unit = trans(#(-1,0)).*matrix(21)|trans(#(1,2))
ind  = nninit (wei, unit)
wei  = index (wei, ind)
wei
```

**Result:** `Content of object wei`

**Group:**
**See also:** nnrsetnet nnrsettrain nnrtest

**Function:** **nnrdovm**

**Description:** nnrdovm optimizes a network for a given dataset.

**Usage:** $\{wtsout, fout\}$ = *nnrdovm(param, wtsin)*

**Input:**

| | | |
|---|---|---|
| | *param* | 3 x 1 vector number of weights, maximal number of iterations and tracing message visible |
| | *wtsin* | vector input weights (either randomly generated or by the user generated). |

**Output:**

| | | |
|---|---|---|
| | *wtsout* | vector output weights after optimization |
| | *fout* | scalar minimal value of error function |

**Result:** `for the use see the macro nnrnet.xpl`

**Group:**
**See also:** nnrsettrain nnrdovm nnrtest

## Function: **nnrsetnet**

**Description:** nnrsetnet sets the internal variables to construct a specific network.

**Usage:** *nnrsetnet (param, nconn, conn)*

**Input:**

| | | |
|---|---|---|
| **Input:** | *param* | 7 x 1 vector |
| | *nconn* | vector codings of connections |
| | *conn* | vector codings of connections |

**Output:**

**Example:**

**Result:** `for the use see the macro nnrnet.xpl`

**Group:**
**See also:** nnrsetnet nnrdovm nnrtest

**Function:** **nnrsettrain**

**Description:** nnrsettrain sets the internal variables to fill a specific network with data and weights.

**Usage:** *nnrtrain (ntr, data, weights)*

**Input:**

| | |
|---|---|
| *ntr* | scalar number of observations |
| *data* | n x (p+q) matrix of input and output data |
| *weights* | n x 1 vector of weights for each observation |

**Output:**

**Result:** `for the use see the macro nnrnet.xpl`

**Group:**
**See also:** nnrsetnet nnrsettrain nnrdovm

## Function: nnrtest

**Description:** nnrtest computes for a given network and dataset the y-values.

**Usage:** *yh = nnrtest (param, data, weights)*

| | | |
|---|---|---|
| **Input:** | *param* | 7 x 1 vector of parameters |
| | *data* | n x (p+q) matrix of input and output data |
| | *weights* | n x 1 vector of weights for each observation |
| **Output:** | *yh* | n x q estimated values |

**Result:** `for the use see the macro nnrpredict.xpl`

**Group:**
**See also:** nninit nnfunc

## Function: nnvisu

**Description:** nnvisu computes the visualization for a given feed forward network by non-metric multi-dimensional scaling.

**Usage:** *pnew = nnfunc (d, pold)*

**Input:**
*d* n x 3 matrix of distances
*pold* m x 2 starting position

**Output:** *pnew* m x 2 computed new positions

**Example:**
```
x = read ("kredit")       ; read kredit data
y = x[1:10,21]
x = x[1:20,1:20]
;; build logit model
wei  = (1:20)~(21.*matrix(20))~normal(20)
unit = trans(#(-1,0)).*matrix(21)|trans(#(1,2))
ind  = nninit (wei, unit)
wei  = index (wei, ind)
randomize(0)
wei[,3] = abs(wei[,3])    ; most intersting connections are near 0
pos  = normal(rows(unit),2)
pos  = nnvisu(wei, pos)
pos
```

**Result:** `Content of object pos`

**Group:** Random Variables
**See also:** cdfn, pdfn, randomize, uniform

## Function: normal

**Description:** normal generates arrays up to eight dimensions of pseudo random variables with a standard normal distribution. the algorithm by box-muller is used.

**Usage:** *z = normal (d1 {, ..., dn})*

**Input:** *d1, ..., dn* scalar (number of elements in dimension 1 ... n)

**Output:** *z* d1 x d2 x ... x dn matrix containing pseudo random variables with a standard normal distribution

**Example:**

```
randomize (0)
normal (2, 3)
```

**Result:**

```
[1,] -0.21293 1.9502 -1.4341
[2,] -1.0078 -1.3052 0.69812
```

**References:** knuth, donald e. 1981 "seminumerical algorithms," 2nd ed. vol 2 of the the art of computer programming

**Group:** Time and Date functions
**Subgroup:** Time as numerical output
**See also:** nyear nmonth ndayofmonth ndayofweek nhour nminute

## Function: nsecond

**Description:** returns the second as a number (0-59)

**Usage:** *tn = nsecond(t)*

**Input:** *t* n x 1 time vector

**Output:** *tn* n x 1 array

**Example:**

```
yr = 1970 + (1:10)
t  = settime(yr)
ts = nsecond(t)
ts
```

**Result:**

```
[ 1,] 0.000000
[ 2,] 0.000000
[ 3,] 0.000000
[ 4,] 0.000000
[ 5,] 0.000000
[ 6,] 0.000000
[ 7,] 0.000000
[ 8,] 0.000000
[ 9,] 0.000000
[10,] 0.000000
```

**Group:** Time and Date functions
**Subgroup:** Time as numerical output
**See also:** nmonth ndayofmonth ndayofweek nhour nminute nsecond

## Function: nyear

**Description:** returns the year as a four digit number (YYYY)

**Usage:** *tn = nyear(t)*

**Input:** *t* n x 1 time vector

**Output:** *tn* n x 1 array

**Example:**

```
yr = 1970 + (1:10)
t  = settime(yr)
ts = nminute(t)
ts
```

**Result:**

```
[ 1,] 1971.000000
[ 2,] 1972.000000
[ 3,] 1973.000000
[ 4,] 1974.000000
[ 5,] 1975.000000
[ 6,] 1976.000000
[ 7,] 1977.000000
[ 8,] 1978.000000
[ 9,] 1979.000000
[10,] 1980.000000
```

**Group:** Operators
**See also:** ||

## Function: &&

**Description:** && is a operator for elementwise logical operation (and). If the condition is false 0 is returned, otherwise 1.

**Usage:** $z = y\ \&\&\ x$

**Input:**

| | | |
|---|---|---|
| | $x$ | array |
| | $y$ | array |

**Output:** $z$ array

**Example:** `(#(3,4,5)<4) && (#(5,4,3)<4)`

**Result:**

```
[1,]     0
[2,]     0
[3,]     0
```

**Group:** Operators
**See also:** // /**/

**Function:** **;**

**Description:** ; is a operator to allow comments in macros and datafiles. All text after this operator is ignored. The Operator // is equvivalent to ;.

**Usage:** *; comments*

**Group:** Operators
**See also:** ; /**/

## Function: //

**Description:** // is a operator to allow comments in macros and datafiles. All text after this operator is ignored. The Operator // is equvivalent to ;.

**Usage:** // *comments*

**Group:** Operators
**See also:** ; //

**Function:** /**/

**Description:** /* */ is a operator to allow comments in macros and datafiles. All text between /* and */ is ignored.

**Usage:** /* *comments* */

**Group:** Operators
**See also:** matrix reduce reshape

**Function:** #

**Description:** # evaluates all arguments as vectors and combines them into a single vector.

**Usage:** *z = #(x1, ... ,xn)*

**Input:** *x1 ... xn* scalar

**Output:** *z* n dimensional column vector containing x1 ... xn

**Notes:** with a single argument, #(x) is equivalent to reshape(x, prod(dim(x))).

**Example:** `#(10:7, 3, 4)'`

**Result:** `[1,]      10   9   8   7   3   4`

**Group:** Operators
**See also:** aseq mseq matrix #

**Function:** **:**

**Description:** : generates additive sequences in ascending or descending order.

**Usage:** $z = l : u$

**Input:**

| | | |
|---|---|---|
| | $l$ | scalar |
| | $y$ | scalar |

**Output:** $z$ (l-u+1) dimensional vector containing l, l+1, ..., u

**Example:** `(4:6)'~(3:1)'`

**Result:**

```
[1,]     4   5   6   3   2   1
```

**Group:** Operators
**See also:** >= <= > <

**Function:** ==

**Description:** == compares elementwise. As usual if the elements are equal 1 is returned otherwise 0.

**Usage:** $z = y == x$

**Input:**

| | | |
|---|---|---|
| | $x$ | array |
| | $y$ | array |

**Output:**

| | | |
|---|---|---|
| | $z$ | array |

**Notes:** z[k1,...,kp] = 1 if and only if only x[k1,...,kp] = y[k1,...,kp].
The operator == is equivalent to .== and .=, which are provided for compatibility reasons with former versions of XploRe.

**Example:**

```
#(3,4) == #(3,2)

#(3,4) .== #(3,4)

#(6,9)' .= #(11,9)'
```

**Result:**

```
[1,]      1
[2,]      0

[1,]      1
[2,]      1

[1,]  0   1
```

**Group:** Operators
**See also:** +, -, .*, *, ^, %,

**Function:** /

**Description:** / is a operator for elementwise division of arrays.

**Usage:** $z = y / x$

**Input:**

| | | |
|---|---|---|
| | $x$ | n1 x n2 x ... x np array |
| | $y$ | m1 x m2 x ... x mp array |

**Output:**

| | | |
|---|---|---|
| | $z$ | max(n1,m1) x max(n2,m2) x ... x max(np,mp) array |

**Notes:** z[k1,k2,....,kp] = x[k1%n1,k2%n2,...,kp%np] / y[k1%m1,k2%m2,...,kp%mp]
The operator / is equivalent to ./ ,which are provided for compatibility reasons with former versions of XploRe.

**Example:** `(#(3,4)~#(1,2))/#(2,3)`

**Result:**

```
[1,]     1.5     0.5
[2,]     1.3333  0.66667
```

**Group:** Operators
**See also:**

**Function:** =

**Description:** = is a operator of assignment for arrays. if the left argument already exists, it will be overwritten. Otherwise it is created. The left side is always an exact copy of the right object.

**Usage:** $y = x$

**Input:** $x$ array

**Output:** $y$ array

**Example:**
```
x = #(1,2)~#(3,4)~#(5,6)
x
```

**Result:**
```
[1,]     1     3     5
[2,]     2     4     6
```

**Group:** Operators
**See also:** + * .* / - %

**Function:** ^

**Description:** ^ is a operator for elementwise exponantiation of arrays.

**Usage:** $z = y\hat{}\,x$

| | | |
|---|---|---|
| **Input:** | $x$ | n1 x n2 x ... x np array |
| | $y$ | m1 x m2 x ... x mp array |
| **Output:** | $z$ | max(n1,m1) x max(n2,m2) x ... x max(np,mp) array |

**Notes:** z[k1,k2,....,kp] = x[k1%n1,k2%n2,...,kp%np] ^ y[k1%m1,k2%m2,...,kp%mp]
The operator ^ is equivalent to .^, which are provided for compatibility reasons with former versions of XploRe.

**Example:**
```
#(3,4) ^ #(1,2)
```

**Result:**
```
[1,]      3
[2,]     16
```

**Group:** Operators
**See also:** > <= == <

## Function: >=

**Description:** >= is a operator for elementwise comparison of arrays (greater or equal). If the condition is false 0 is returned, otherwise 1.

**Usage:** $z = y >= x$

**Input:**

| | | |
|---|---|---|
| | $x$ | array |
| | $y$ | array |

**Output:**

| | | |
|---|---|---|
| | $z$ | array |

**Notes:** z[k1,...,kp] = 1 if and only if only x[k1,...,kp] >= y[k1,...,kp].
The operator >= is equivalent to .>=, which are provided for compatibility reasons with former versions of XploRe.

**Example:**

```
#(3,4) >= #(3,2)

#(3,4) .>= #(3,4)
```

**Result:**

```
[1,]     1
[2,]     1

[1,]     1
[2,]     1
```

**Group:** Operators
**See also:** >= <= == <

**Function:** >

**Description:** > is a operator for elementwise comparison of arrays (greater). If the condition is false 0 is returned, otherwise 1.

**Usage:** $z = y > x$

**Input:**

| | | |
|---|---|---|
| | $x$ | array |
| | $y$ | array |

**Output:**

| | | |
|---|---|---|
| | $z$ | array |

**Notes:** z[k1,...,kp] = 1 if and only if only x[k1,...,kp] > y[k1,...,kp].
The operator > is equivalent to .>, which are provided for compatibility reasons with former versions of XploRe.

**Example:**

```
#(3,4) > #(3,2)

#(3,4) .> #(3,4)
```

**Result:**

```
[1,]     0
[2,]     1

[1,]     0
[2,]     0
```

**Group:** Operators
**See also:** | stack

**Function:** $\sim$

**Description:** $\sim$ combines arrays columnwise.

**Usage:** $z = x \sim y$

**Input:** $x$ array
$y$ array

**Output:** $z$ array

**Example:** `matrix(2,3) ~ #(1, 2) ~ #(3, 4)`

**Result:**
```
[1,]     1   1   1   1   3
[2,]     1   1   1   2   4
```

**Group:** Operators
**See also:** > <= == >=

**Function:** <

**Description:** < is a operator for elementwise comparison of arrays (smaller). If the condition is false 0 is returned, otherwise 1.

**Usage:** $z = y < x$

**Input:**

| | | |
|---|---|---|
| | $x$ | array |
| | $y$ | array |

**Output:**

| | | |
|---|---|---|
| | $z$ | array |

**Notes:** z[k1,...,kp] = 1 if and only if only x[k1,...,kp] < y[k1,...,kp].
The operator < is equivalent to .<, which are provided for compatibility reasons with former versions of XploRe.

**Example:**

```
#(3,4) < #(3,2)

#(3,4) .< #(3,4)
```

**Result:**

```
[1,]     0
[2,]     0

[1,]     0
[2,]     0
```

**Group:** Operators
**See also:** > < == >=

## Function: <=

**Description:** <= is a operator for elementwise comparison of arrays (smaller or equal). If the condition is false 0 is returned, otherwise 1.

**Usage:** $z = y <= x$

**Input:**

| | |
|---|---|
| $x$ | array |
| $y$ | array |

**Output:**

| | |
|---|---|
| $z$ | array |

**Notes:** z[k1,...,kp] = 1 if and only if only x[k1,...,kp] <= y[k1,...,kp].
The operator <= is equivalent to .<=, which are provided for compatibility reasons with former versions of XploRe.

**Example:**

```
#(3,4) <= #(3,2)

#(3,4) .<= #(3,4)
```

**Result:**

```
[1,]     1
[2,]     0

[1,]     1
[2,]     1
```

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** [], dot, list, size

## Function: {}

**Description:** {} gives an element of a list.

**Usage:** $y = x\{n\}$

**Input:**

| | |
|---|---|
| $x$ | list |
| $n$ | integer, number of an element of x |

**Output:**

| | |
|---|---|
| $y$ | element of x with number n |

**Notes:** An Element of a list can be an array or a list.
Another kind to select an element is with the name and the dot-operator.

**Example:**

```
x1=#(1,3)~#(3,1)
x2=4:6
y=list(x1,x2)
y{1}
```

**Result:**

```
[1,]     1     3
[2,]     3     1
```

**Group:** Operators
**See also:** + - .* / ^ %

**Function:** *

**Description:** * computes the matrix multiplication of arrays.

**Usage:** $z = y * x$

**Input:**

| | | |
|---|---|---|
| | $x$ | array |
| | $y$ | array |

**Output:**

| | | |
|---|---|---|
| | $z$ | array |

**Notes:** if one or both arguments are scalar, then the usual elementwise multiplication is computed. Otherwise the number of cols of x has to be equal to the number of rows of y.

**Example:** `(#(3,4)~#(1,2))*#(2,3)`

**Result:**

```
[1,]     9
[2,]    14
```

**Group:** Operators
**See also:** + * .* / ^ %

**Function:** -

**Description:** - is a operator for subtraction of arrays.

**Usage:** $z$ = $y$ - $x$

**Input:**

| | |
|---|---|
| $x$ | n1 x n2 x ... x np array |
| $y$ | m1 x m2 x ... x mp array |

**Output:**

| | |
|---|---|
| $z$ | max(n1,m1) x max(n2,m2) x ... x max(np,mp) array |

**Notes:** z[k1,k2,....,kp] = x[k1%n1,k2%n2,...,kp%np] - y[k1%m1,k2%m2,...,kp%mp]
The operator - is equivalent to .-, which are provided for compatibility reasons with former versions of XploRe.

**Example:**

```
#(3,4) - #(1,2)

(#(3,4)~#(1,2)) .- #(2,3)
```

**Result:**

```
[1,]     2
[2,]     2

[1,]     1    -1
[2,]     1    -1
```

**Group:** Operators
**See also:** +, -, .*, *, ^, /,

**Function:** %

**Description:** % is a operator for modulo division of arrays of matrices.

**Usage:** $z = y \% x$

| | | |
|---|---|---|
| **Input:** | $x$ | n1 x n2 x ... x np array |
| | $y$ | m1 x m2 x ... x mp array |
| **Output:** | $z$ | max(n1,m1) x max(n2,m2) x ... x max(np,mp) array |

**Notes:** z[k1,k2,....,kp] = x[k1%n1,k2%n2,...,kp%np] % y[k1%m1,k2%m2,...,kp%mp]

**Example:** `(#(3,4)~#(1,2))%#(2,3)`

**Result:**
```
[1,]     1     1
[2,]     1     2
```

**Group:** Operators
**See also:** + * / - ^ %

## Function: .*

**Description:** .* is a operator for elementwise multiplication of arrays.

**Usage:** $z = y$ .* $x$

**Input:**

| | | |
|---|---|---|
| | $x$ | n1 x n2 x ... x np array |
| | $y$ | m1 x m2 x ... x mp array |

**Output:**

| | | |
|---|---|---|
| | $z$ | max(n1,m1) x max(n2,m2) x ... x max(np,mp) array |

**Notes:** z[k1,k2,....,kp] = x[k1%n1,k2%n2,...,kp%np] .* y[k1%m1,k2%m2,...,kp%mp]

**Example:**

```
#(3,4) .* #(1,2)
```

**Result:**

```
[1,]      3
[2,]      8
```

**Group:** Operators
**See also:**

**Function:** _

**Description:** this operator evaluates the following string as function or macro name.

**Usage:** _*name*

**Input:** *name* text containing the name

**Output:** depends on the executed function or macro

**Notes:** temporary text objects are not allowed

**Example:**
```
fun = "sqrt"
_fun(9)
```

**Result:** `[1,]     3.000000`

**Group:** Operators
**See also:** || &&

**Function:** **!**

**Description:** ! is a operator for elementwise logical operation (not).

**Usage:** $z = ! x$

**Input:** $x$ array

**Output:** $z$ array

**Example:** `(! (#(4,3,2) < 4))'`

**Result:** `[1,]     1  0  0`

**Group:** Operators
**See also:** >= <= == < >

## Function: !=

**Description:** != compares elementwise. As usual if the elements are not equal 1 is returned otherwise 0.

**Usage:** $z = y \mathbin{!=} x$

**Input:**

| | | |
|---|---|---|
| | $x$ | array |
| | $y$ | array |

**Output:**

| | | |
|---|---|---|
| | $z$ | array |

**Notes:** z[k1,...,kp] = 1 if and only if only x[k1,...,kp] != y[k1,...,kp].
The operator != is equivalent to .<> and <>, which are provided for compatibility reasons with former versions of XploRe.

**Example:**

```
#(3,4) != #(3,2)

#(3,4) <> #(3,4)
```

**Result:**

```
[1,]     0
[2,]     1

[1,]     0
[2,]     0
```

**Group:** Operators
**See also:** &&

**Function:** ||

**Description:** || is a operator for the logical operation or.

**Usage:** $z = x \,||\, y$

**Input:**

| | | |
|---|---|---|
| | $x$ | array |
| | $y$ | array |

**Output:**

| | | |
|---|---|---|
| | $z$ | array |

**Example:** `(#(3,4,5)<4) || (#(5,4,3)<4)`

**Result:**

```
[1,]      1
[2,]      0
[3,]      1
```

**Group:** Operators
**See also:** * .* / - ^ %

## Function: +

**Description:** + is a operator for addition of arrays.

**Usage:** $z = y + x$

**Input:**

| | |
|---|---|
| $x$ | n1 x n2 x ... x np array |
| $y$ | m1 x m2 x ... x mp array |

**Output:**

| | |
|---|---|
| $z$ | max(n1,m1) x max(n2,m2) x ... x max(np,mp) array |

**Notes:** z[k1,k2,....,kp] = x[k1%n1,k2%n2,...,kp%np] + y[k1%m1,k2%m2,...,kp%mp]
The operator + is equivalent to .+, which are provided for compatibility reasons with former versions of XploRe.

**Example:** `#(3,4) + #(1,2)`

**Result:**

```
[1,]     4
[2,]     6
```

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** dot, paf, {}

## Function: []

**Description:** [] gives an subarray of an array.

**Usage:** *y = x[{,...,} d1 {,d2{,...}}]*

**Input:**

| | |
|---|---|
| *x* | multidimensional array |
| *di i=1,2,...* | vector |

**Output:**

| | |
|---|---|
| *y* | subarray of x |

**Example:**

```
x=1:12
y=reshape(x,#(3,4))
y[2:3,1:3]
```

**Result:**

```
[1,]     2     5     8
[2,]     3     6     9
```

**Group:** Linear Algebra
**See also:** trans, inv

**Function:** **'**

**Description:** ' transposes rows and columns of an array. This operator is equal to the function trans.

**Usage:** $y = x$'

**Input:** $x$ array

**Output:** $y$ array

**Example:**
```
x = #(1,2)~#(3,4)~#(5,6)
x'
```

**Result:**
```
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

**Group:** Operators
**See also:** ~ stack

**Function:** |

**Description:** | combines arrays rowwise.

**Usage:** $z = x \mid y$

**Input:**

| | | |
|---|---|---|
| | $x$ | n x p matrix |
| | $y$ | m x p matrix |

**Output:** $z$ (n+m) x p matrix

**Example:** `matrix(2,2) | #(1, 2)' | #(3, 4)'`

**Result:**

```
[1,]     1   1
[2,]     1   1
[3,]     1   2
[4,]     3   4
```

**Group:** Reading and Writing
**See also:** write, read

## Function: **output**

**Description:** writes results which are printed into the output window simultaneously into a specified file.

**Usage:** *output (fname, option)*

**Input:**

| | |
|---|---|
| *fname* | string, name of the file which should be used |
| *option* | string, defines the operation "reset" file will be generated or overwritten respectively, "append" output will be appended or generated, "close" file is closed, |

**Notes:** Only a single file can be used at the same time. If you open a new file without closing the old one, the latter is automatically closed.

**Example:**

```
output ("xout", "reset")
pi
output ("xout", "close")
```

**Result:** `Result:   the file 'xout' in the XploRe home directory is generated.`

**Group:** Matrix Manipulation
**Subgroup:** General Commands
**See also:** [], index, matrix

## Function: paf

**Description:** paf selects rows of an object depending on a given vector. All elements of the first argument are taken if the corresponding vector element is unequal to 0. Otherwise there are dropped.

**Usage:** *z = paf (x, ind)*

**Input:**

| | |
|---|---|
| *x* | d1 x ... x dn array |
| *ind* | d1 x 1 vector |

**Output:**

| | |
|---|---|
| *z* | k x d2 x ... x dn array where k=sum_i ind_i |

**Example:**

```
randomize(0)
x=uniform(10)
paf(x, x>0.5)
```

**Result:**

```
[1,] 0.60488
[2,] 0.61585
```

**Group:** Statistical Distributions
**See also:** cdfb, cdfc, cdff, cdfn, cdft, pdfc, pdff, pdfn, pdft, qfc, qff, qfn, qft

## Function: pdfb

**Description:** pdfb returns the value of the density of the beta distribution function for elements of an array.

**Usage:** *z = pdfb (x,a,b)*

| | | |
|---|---|---|
| **Input:** | *x* | n x p x d1 x d2 x ... x dn array |
| | a | n x p x d1 x d2 x ... x dn array or scalar |
| | *b* | n x p x d1 x d2 x ... x dn array or scalar |

**Output:** *z* n x p x d1 x d2 x ... x dn array

**Example:** `pdfb (0.5, 1.0,  1.0)`

**Result:** `1`

**Group:** Statistical Distributions
**See also:** cdfb, cdfc, cdff, cdfn, cdft, pdfb, pdff, pdfn, pdft, qfc, qff, qfn, qft

## Function: pdfc

**Description:** pdfc returns the value of the density of the chi-square-distribution with d degrees of freedom for elements of an array.

**Usage:** *z = pdfc (x, d)*

**Input:**
*x* n x p x d1 x d2 x ... x dn array
*d* n x p x d1 x d2 x ... x dn array or scalar

**Output:** *z* n x p x d1 x d2 x ... x dn array

**Notes:** For values x <=0 missing values are generated.

**Example:** `pdfc (12.591, 6)`

**Result:** `0.018278`

**Group:** Statistical Distributions
**See also:** cdfb, cdfc, cdff, cdfn, cdft, pdfb, pdfc, pdfn, pdft, qfc, qff, qfn, qft

## Function: pdff

**Description:** pdff returns the value of the density of the F-distribution with d and e degrees of freedom for elements of an array.

**Usage:** *z = pdff (x, d, e)*

**Input:**

| | | |
|---|---|---|
| | *x* | n x p x d1 x d2 x ... x dn array |
| | *d* | n x p x d1 x d2 x ... x dn array or scalar |
| | *e* | n x p x d1 x d2 x ... x dn array or scalar |

**Output:** *z* n x p x d1 x d2 x ... x dn array

**Notes:** For values x < 0 missing values are generated.

**Example:** `pdff (5.41, 3, 5)`

**Result:** `0.016939`

**Group:** Statistical Distributions
**See also:** cdfb, cdfc, cdff, cdfn, cdft, normal, pdfb, pdfc, pdff, pdft, qfc, qff, qfn, qft

## Function: pdfn

**Description:** pdfn returns the value of the density of the standard normal distribution for elements of an array.

**Usage:** *z = pdfn (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Example:** `pdfn (1.96)`

**Result:** `[1,] 0.058441`

**Group:** Statistical Distributions
**See also:** cdfb, cdfc, cdff, cdfn, cdft, pdfb, pdfc, pdff, pdfn, qfc, qff, qfn, qft

## Function: pdft

**Description:** pdft returns the value of the density of the t-distribution with d degrees of freedom for elements of an array.

**Usage:** *z = pdft (x, d)*

| | | |
|---|---|---|
| **Input:** | *x* | n x p x d1 x d2 x ... x dn array |
| | *d* | n x p x d1 x d2 x ... x dn array or scalar |
| **Output:** | *z* | n x p x d1 x d2 x ... x dn array |

**Example:** `pdft (2.92, 16 )`

**Result:** `0.010405`

**Group:** Plot Manipulation
**See also:** createdisplay, show

## Function: print

**Description:** print a display or a datapart of it.

**Usage:** *print(di{, l_num, c_num} {, filename})*

**Input:**

| | |
|---|---|
| *di* | Display |
| *l_num* | integer, 1 ... count of lines of di |
| *c_num* | integer, 1 ... count of columns of di |
| *filename* | string, a name of a ps-file |

**Example:**

```
di=createdisplay(1, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
show(di, 1, 1, x~y, x~knn(y, 10))
print(di, "TEST.ps")
```

**Result:** `A file named "TEST.ps" is created and with the content from the display di filled.`

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** cumprod, cumsum, sum

**Function:** **prod**

**Description:** prod computes the product of the elements of an array regarding a given dimension.

**Usage:** *z = prod (x {,j})*

**Input:**
*x* d1 x d2 x .. x dn array
*j* integer (number of dimension, default = 1)

**Output:**
*z* d1 x d2 x .. x d(j-1) x 1 x d(j+1) x .. x dn array

**Example:**

```
x = #(1, 2)~#(3, 4)~#(5, 6)
prod (x,2)
```

**Result:**

```
[1,]    15
[2,]    48
```

**Group:** Information and Control
**See also:** existglobal, getglobal

**Function:** **putglobal**

**Description:** putglobal generates a global variable

**Usage:** *putglobal (x)*

**Input:** *x* name of an arbitrary object

**Notes:** This command is only useful within macros

**Example:**
```
x = 7
putglobal ("x")
```

**Result:** `x is now a global variable containing 7`

**Group:** Statistical Distributions
**See also:** cdfb, cdfc, cdff, cdfn, cdft, pdfb, pdfc, pdff, pdfn, pdft, qff,qfn, qft

## Function: qfc

**Description:** qfc returns the inverse of a chi-square-distribution function with d degrees of freedom for elements of an array.

**Usage:** *y = qfc (x, d )*

**Input:** *x* n x p x d1 x d2 x ... x dn array
*d* n x p x d1 x d2 x ... x dn array or scalar

**Output:** *y* n x p x d1 x d2 x ... x dn array

**Notes:** For elements of x that do not belong to [0,1] NaN values are generated.

**Example:** `qfc (0.95,6)`

**Result:** `12.59`

**Group:** Statistical Distributions
**See also:** cdfb, cdfc, cdff, cdfn, cdft, pdfb, pdfc, pdff, pdfn, pdft, qfc, qfn, qft

## Function: qff

**Description:** qff returns the inverse of a F-distribution function with d1 and d2 degrees of freedom for elements of an array.

**Usage:** *y = qff (x, d1, d2)*

| | | |
|---|---|---|
| **Input:** | *x* | n x p x d1 x d2 x ... x dn array |
| | *d1* | n x p x d1 x d2 x ... x dn array or scalar |
| | *d2* | n x p x d1 x d2 x ... x dn array or scalar |
| **Output:** | *y* | n x p x d1 x d2 x ... x dn array |

**Notes:** For elements of x that do not belong to [0,1] NaN values are generated.

**Example:** `qff (0.95, 3, 5)`

**Result:** `5.41`

**Group:** Statistical Distributions
**See also:** cdfb, cdfc, cdff, cdfn, cdft, pdfb, pdfc, pdff, pdfn, pdft, qfc, qff, qft

## Function: qfn

**Description:** qfn returns the inverse of a standard normal distribution function for elements of an array.

**Usage:** *y = qfn (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *y* d1 x d2 x ... x dn array

**Notes:** for elements of x that do not belong to [0,1] NaN values are generated.

**Example:** `qfn (#(0.5987, 1.2)')`

**Result:** `[1,] 0.24998   NaN`

**Group:** Statistical Distributions
**See also:** cdfb, cdfc, cdff, cdfn, cdft, pdfb, pdfc, pdff, pdfn, pdft, qfc, qff, qfn

## Function: qft

**Description:** qft returns the inverse of a t - distribution function with d degrees of freedom for elements of an array.

**Usage:** *y = qft (x, d)*

**Input:**

| | |
|---|---|
| *x* | n x p x d1 x d2 x ... x dn array |
| *d* | n x p x d1 x d2 x ... x dn array |

**Output:**

| | |
|---|---|
| *y* | n x p x d1 x d2 x ... x dn array |

**Notes:** For elements of x that do not belong to [0,1] NaN values are generated.

**Example:** `qft (0.995, 16)`

**Result:** `2.9221`

**Group:** Random Variables
**See also:** normal, uniform

## Function: randomize

**Description:** randomize sets the seed of the pseudonumber generator. when xplore starts it calls randomize(0).

**Usage:** *randomize (s)*

**Input:** *s* scalar

**Example:** `randomize (101)`

**Group:** Reading and Writing
**See also:** write, readm

## Function: read

**Description:** read is a command to read data from a file. Each column of the file will be interpreted as a vector of numbers.

**Usage:** *z = read (file)*

**Input:** *file* string with the name of the file to be read

**Output:** *z* matrix

**Notes:** If an element in the file is not a number, read will return 0 for this element.

**Example:**
```
write(reshape(1:15,#(3,5)),"my_data.dat")
read("my_data.dat")
```

**Result:**
```
[1,]     1     4     7    10    13
[2,]     2     5     8    11    14
[3,]     3     6     9    12    15
```

**Group:** Reading and Writing
**See also:** write, read, readm

## Function: readascii

**Description:** readascii is a command to read ASCII data from a file.

**Usage:** {*data*, *type*} = *readascii (filename, separator* {, *check*})

**Input:**

| | | |
|---|---|---|
| | *filename* | string with the name of the file to be read |
| | *separator* | string with field separators |
| | *check* | determine type (default 1) |

**Output:**

| | | |
|---|---|---|
| | *data* | n x 1 string(!) vector |
| | *type* | n x 1 numeric vector |

**Notes:** Each char in the separator string is a possible separator for a data field in a file. The standard separator string should be " \t\n" (blank, tab, newline). For example EXCEL data can be saved in a format such that the data fields separated by semicolon. In such a case the separator string might be " \t\n;". If one of the separators is the slash then use "\\". If check is one (default) you get back in w.type the type of the data determined by XploRe. If check is zero the w.type is always 10 (text) except for the newline.

The types returned are: 0 (number), 1 (missing), 10 (text), 20 (newline).

**Example:**

```
w = readascii ("geyser", " \t\n")
w
```

**Result:**

```
Content of object w.data

[  1,] 0.360000E+01
[  2,] 79.
[  3,]

[  4,] 0.180000E+01
[  5,] 54.
[  6,]
...
[814,] 0.446667E+01
[815,] 74.
[816,]

Content of object w.type

[  1,]        0
[  2,]        0
[  3,]       20
[  4,]        0
[  5,]        0
[  6,]       20
...
[814,]        0
[815,]        0
[816,]       20
```

**Example:**

```
w = readascii ("geyser", " \t\n", 0)
w
```

**Result:**

```
Content of object w.data

[  1,] 0.360000E+01
[  2,] 79.
[  3,]

[  4,] 0.180000E+01
[  5,] 54.
[  6,]
...
[814,] 0.446667E+01
[815,] 74.
[816,]

Content of object w.type

[  1,] 10
[  2,] 10
[  3,] 20
[  4,] 10
[  5,] 10
[  6,] 20
...
[814,] 10
[815,] 10
[816,] 20
```

**Group:** Statistical Data Analysis
**Subgroup:** Graphical User Interfac
**See also:** setmode

## Function: readevent

**Description:** readevent reads a key- or a mouse- event while a program is running. An "event" is a stroke of a key or a click of a mouse button. readevent will be mainly useful for letting XploRe know whether such an event has occured and to get some special information like the coordinates where the mouse click happened or a key code. readevent will "record" the relevant event if it ocurred previous to the moment when readevent is called. readevent can therefore be used to "tell" the program that the event has happened. Sometimes it is usefull to call at first setmode(...., 2) to disable default event handling.

**Usage:** *ret_event=readevent(which_event{, di, row_num, col_num})*

**Input:**

| | |
|---|---|
| *which_event* | 0 for key; 1 for mouse |
| *di* | display |
| *row_num* | integer |
| *col_num* | integer |

**Output:**

| | |
|---|---|
| *ret_event* | a complex object:<br>1, if this event happened<br>0, if this event didn't happen<br>ret_event can include following elements: |
| *ret_event.key* | the string of the key if a key has been pressed |
| *ret_event.keysym* | the integer code of the key if a key has been pressed (the mapping of keys to integer codes may vary with the operating system) |
| *ret_event.button* | one of the integers 1,2,3, returning which mouse button has been pressed |
| *ret_event.xy* | (only if called with 4 arguments)<br>a (1x2) vector of integers, returning the window coordinates of the mouse pointer if a mouse button has been pressed (e.g. 0∼0 is the upper left corner of the window) |
| *ret_event.xyreal* | (only if called with 4 arguments and the plotted data has 2 dimensions)<br>a (1x2) integer vector returning the real coordinates of the mouse pointer in the data space. |

**Example:**

```
d=createdisplay(1, 1)
x=(1:500)*6.28/50
u=sin(x)~sin(3*x)~(sin(2*x)/100)~sin(5*x)~(sin(7*x)/100)
setmaskl(u, (1:rows(u))', 1, 1, 1)
show(d, 1, 1, u)
proc (m) = em(dim)
i=0
m=matrix(dim, dim)
m=m-m
while (i<dim)
i=i+1
m[i+1, i+1]=1
endo
endp
proc () = rotate (d, x, y)
keypressed=readevent(0)
while (keypressed==0)
go=getgopt(d, x, y)
dim=cols(go.rotcos)
angle=uniform(dim, 1)/5
i=1
m=em(dim)
while (i<dim)
i=i+1
mm=em(dim)
mm[1, 1]=cos(angle[i])
mm[i, i]=cos(angle[i])
mm[1, i]=-sin(angle[i])
mm[i, 1]=sin(angle[i])
m=m*mm
endo
gort=go.rotcos
gort=gort*m
setgopt(d, x, y, "rotcos", gort)
keypressed=readevent(0)
endo
endp
rotate(d, 1, 1)
```

**Result:** a line that you see is rotated until you press some key.

**Group:** Reading and Writing
**See also:** write, read

## Function: readm

**Description:** readm is a command to read mixed data from a file.

**Usage:** *z = readm (file)*

| | | |
|---|---|---|
| **Input:** | *file* | string with the name of the file to be read |
| **Output:** | *z.double* | n x p matrix float vectors |
| | *z.text* | n x q matrix text vectors |
| | *z.integer* | n x r matrix integer vectors |
| | *z.type* | (p+q+r) vector types of columns in file |

**Notes:** If you read mixed data from a file the original position of the vector in the file gets lost. Thus you will receive 3 matrices which will contain the vectors of one type. It is ensured that, e.g. z.double[,1] was in the file before z.double[,2]. The last parameter z.type contains the recognized type t of the corresponding vector in the file. A 1 represents text, a 2 an integer and a 3 a float vector. The file "mixed.dat" contains the following vectors:

1 2 3 4 hallo 4.5 .
5 6 7.0 0 "hallo again" 6.7 8.9

The first two vector will be recognized as integer, the next vector will be a float vector because of the 7.0. The fourth vector is again an integer vector. The fifth vector is a text vector; if the text is enclosed in apostrophs blanks can be incorporated in the text. The last two vectors are float vectors. The single dot will be interpreted as NAN.

**Example:**

```
w = read ("mixed.dat")
w
```

**Result:**

```
Content of object w.double

[1,]        3      4.5      NaN
[2,]        7      6.7      8.9
Content of object w.text

[1,] hallo
[2,] hallo again
Content of object w.integer

[1,]        1        2        4
[2,]        5        6        0
Content of object w.type

[1,]        2
[2,]        2
[3,]        3
[4,]        2
[5,]        1
[6,]        3
[7,]        3
```

**Group:** GUI functions
**See also:** selectitem

## Function: **readvalue**

**Description:** readvalue reads one or many values

**Usage:** *v=readvalue(valuestrs, values)*

**Input:**

| | | |
|---|---|---|
| | *valuestrs* | a vector of strings to explain the values |
| | *values* | a vector of default values |

**Output:** *v* a vector of values given by user

**Example:**

```
valuestrs="x=" | "y=" | "z="
values=0 | 0 | 0
v=readvalue(valuestrs, values)
```

**Result:** `a box is opened where you schould put your values. After pressing <OK> a vector v containing the values from this box will be created.`

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** {}, reduce, vec

## Function: reshape

**Description:** reshape transforms an array into a new one with given dimensions.

**Usage:** *y = reshape(x,d)*

**Input:**

| | | |
|---|---|---|
| **Input:** | *x* | multidimensional array (or string) |
| | *d* | vector containing the new dimensions |

**Output:**

| | | |
|---|---|---|
| **Output:** | *y* | array, witch has the dimension d |

**Notes:** reshape can not change the number of elements of an array. That means, the number of elements of x is equal to that one of y. Actually this is prod (d).
To save memory it is possible to reshape an object itself by giving the name of the object, e.g., reshape ("x", #(2,5,2)). Then reshape has no result.

**Example:**

```
x=1:20
reshape(x,#(2,5,2))
```

**Result:**

```
[,,1,1,1,1,1,1]
[1,]     1     3     5     7     9
[2,]     2     4     6     8    10

[,,2,1,1,1,1,1]
[1,]    11    13    15    17    19
[2,]    12    14    16    18    20
```

**Group:** Statistical Data Analysis
**Subgroup:** Nonparametric Methods
**See also:** sker isoreg l1line lowess

**Function:** **rmed**

**Description:** rmed computes the running median of y using the optimal median smoothing algorithm od Haerdle ans Steiger (1990).

**Usage:** *yh = rmed (y, k)*

**Input:**

| | |
|---|---|
| *y* | n x 1 vector |
| *k* | scalar |

**Output:**

| | |
|---|---|
| *yh* | n x 1 vector |

**Notes:** k has to be odd

**Example:**

```
; generate a dataset
x = normal (100)
; running median with window-width 11
z = rmed (x, 11)
```

**Result:** `z contains the running median  fit of x.`

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** cols, dim

## Function: rows

**Description:** rows returns the number of rows in an array.

**Usage:** *z = rows (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* scalar with the number d1

**Example:** `rows (matrix (10, 2))`

**Result:** `[1,]    10`

**Group:** Reading and Writing
**See also:** editmac, write

**Function:** **savemac**

**Description:** savemac saves the given macro into a file

**Usage:** *savemac (macro_name {, file})*

**Input:**

| | | |
|---|---|---|
| | *macro_name* | string with the name of the macro |
| | *file* | string with the name of the file |

**Notes:** If no filename is given, savemac will write macro_name into the file "macro_name.xpl".
If no path is specified, the file will be written into your XploRe directory.

**Example:**

```
proc(j)=factorial(x)
prod(1:x)
endp

savemac("factorial")
```

**Result:** `Writes the macro factorial into the file factorial.xpl in your XploRe directory.`

**Group:** Time and Date functions
**Subgroup:** Time as string output
**See also:** year month dayofmonth dayofweek hour minute

## Function: second

**Description:** returns the second as a string

**Usage:** *ts = second(t)*

**Input:** *t* n x 1 time vector

**Output:** *ts* n x 1 string array

**Example:**

```
yr = 1970 + (1:10)
t  = settime(yr)
ts = second(t)
ts
```

**Result:**

```
[ 1,] 00
[ 2,] 00
[ 3,] 00
[ 4,] 00
[ 5,] 00
[ 6,] 00
[ 7,] 00
[ 8,] 00
[ 9,] 00
[10,] 00
```

**Group:** GUI functions
**See also:** readvalue

## Function: selectitem

**Description:** selectitem makes a menu defined by user

**Usage:** *i=selectitem(headline, items)*

| | | |
|---|---|---|
| **Input:** | *headline* | a string |
| | *items* | a vector of strings |
| **Output:** | *i* | a number of selected item, 1 ... rows of items |

**Example:**

```
headline="Please select"
items="linear" | "quadratic" | "log"
i=selectitem(headline, items)
```

**Result:** `a menu with this content is created, i contents the number of item you have selected.`

**Group:** Information and Control
**See also:** getenv, library

**Function:** **setenv**

**Description:** setenv sets one of the environment variables (xpl4home, xpl4data, xpl4help, xpl4prog, xpl4backup, format) format sets the output format used by the printf of the C library.

**Usage:** *setenv (envname, newval)*

**Input:**

*envname* name of the environment "xpl4home" XploRe home directory (is better set by invoking XploRe) "xpl4data" standard directory for loading data "xpl4lib" standard directory for loading librarys "xpl4dll" directory, used by dlopen, to load a dynamically linked library "xpl4help" used by starting the HTML-browser "xpl4prog" standard directory for the program file selection "xpl4backup" directory to back up program files "format" controls the numerical output. Syntax like C's printf "stringformat" controls the textual output. Syntax like C's printf "browser" sets the HTML-browser for the help system

*newval* new value (e.g., name of a directory)

**Notes:** It is possible to set the enviroment variables for xpl4lib, xpl4dll and xpl4data to more than one directory. In this case, pathnames have to be seperated by :.

**Example:**

```
setenv ("xpl4data", "/user/haerdle/project1/data")
setenv ("format", "%10.2f")
```

**Result:**

```
data are read from /user/haerdle/project1/data
numbers are printed rounded to two digits
```

**Group:** Plot Manipulation
**See also:** createdisplay

## Function: **setfractions**

**Description:** changes proportions of parts of display

**Usage:** *setfractions(di, rows_props, cols_props)*

**Input:**

| | |
|---|---|
| *di* | Display |
| *rows_props* | vector of values for vertical fractions, by #(1, 1, 1) all parts become the same height |
| *cols_props* | vector of values for horizontal fractions, by #(1, 1, 1) all parts become the same width |

**Example:**

```
di=createdisplay(2, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
setfractions(di, 5 | 1, 1)
show(di, 2, 1, "y=sin(x/20)+uniform(100, 1)/10")
show(di, 1, 1, x~y)
```

**Result:** `you see a big window with a  dataset and a small window with a string.`

**Group:** Plot Manipulation
**See also:** show, getgopt

## Function: setgopt

**Description:** setgopt sets the layout of a plot. There are two kinds of this command: with a special parameter that is get by command getgopt(), or with some pairs for different options.

**Usage:** *setgopt(di, l_num, c_num, gropt) or setgopt(di, l_num, c_num, optionname, optionvalue {, optionname1, optionvalue1} {, optionname2, optionvalue2} {...})*

**Input:**

- *di* Display
- *l_num* integer, 1 ... count of lines of di
- *c_num* integer, 1 ... count of columns of di
- *gropt* a special object, taht is get by the command gropt=getgopt(...) and uncludes all grafic options from another plot.
- *optionname* the option, that is to change. Allowed are: "title" to change headline, "xlim" to change limits for x-axis, "ylim" to change limits for y-axis, "xoffset" to change the width of axis border "yoffset" to change the height of axis border "xvalue" to change the values m and k by m+k*x, "yvalue" to change the values m and k by m+k*y, "xorigin" to change origin for tickmark for x-axis, "yorigin" to change origin for tickmark for y-axis, "xmajor" to change major for tickmark for x-axis, "ymajor" to change major for tickmark for y-axis, "xlabel" to change label for x-axis, "ylabel" to change label for y-axis, "rotpoint" to change rotation point , "rotcos" to change rotation cosinus matrix, "scal" to change scale matrix, "transl" to change translation vector
- *optionvalue* the new value for this option "title" a string "xlim" a vector that contents two values, f. e. 0 | 10 "ylim" a vector that contents two values, f. e. 0 | 10 "xoffset" a vector that contents two values to change the right and left widths of axis border (in percents) "yoffset" a vector that contents two values to change the upper an d lower heights of axis border (in percents) "xvalue" a vector that contents two values to change the values m and k by m+k*x, f. g. 0 | 1 for 0+1*x "yvalue" a vector that contents two values to change the values m and k by m+k*y, f. g. 0 | 1 for 0+1*y "xorigin" a value ( usually 0.0) "yorigin" a value ( usually 0.0) "xmajor" a value for major for tickmark for x-axis "ymajor" a value for major for tickmark for y-axis "xlabel" a string for label for x-axis "ylabel" a string for label for y-axis "rotpoint" a rotation point vector "rotcos" a rotation cosinus matrix "scal" a diagonal scale matrix "transl" a translation vector

**Example:**

```
di=createdisplay(4, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
show(di, 1, 1, x~y)
show(di, 2, 1, x~y)
show(di, 3, 1, x~y)
show(di, 4, 1, x~y)
;  you see that thanks different width and size of each plot the axisborder have different width and height too. Futhermore, the height of border is not enough to show the axis values and headline. We try to improve it:
setgopt(di, 1, 1, "yoffset", 20|20, "xoffset", 5|5)
setgopt(di, 2, 1, "yoffset", 20|20, "xoffset", 5|5)
setgopt(di, 3, 1, "yoffset", 20|20, "xoffset", 5|5)
setgopt(di, 4, 1, "yoffset", 20|20, "xoffset", 5|5)
```

**Result:** `you see it yourself.`

**Group:** Plot Manipulation
**See also:** setyaxis, setxaxis, show,

## Function: setheadline

**Description:** setheadline changes the head line of plot.

**Usage:** *setheadline(di, l_num, c_num, headline)*

**Input:**

| | |
|---|---|
| *di* | Display |
| *l_num* | integer, 1 ... count of lines of di |
| *c_num* | integer, 1 ... count of columns of di |
| *headline* | a string |

**Example:**

```
di=createdisplay(1, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
show(di, 1, 1, x~y)
setheadline(di, 1, 1, "A test")
```

**Result:** `you see it yourself.`

**Group:** Plot Manipulation
**See also:** setmaskp, setmaskt, show

**Function:** **setmaskl**

**Description:** connects points of a dataset that is already plotted.

**Usage:** *setmaskl(data, p_matrix, color_num, art, thick)*

**Input:**

| | |
|---|---|
| *data* | a data matrix |
| *p_matrix* | a matrix which contains as rows the vectors from points that are to be connected. The elements of p_matrix to be min. 1 and max. count of points of data; 0 as element from it breaks the line. |
| *color_num* | a vector of integers, 0 ... 7. The dimension of color_num must be equal to count of rows from p_matrix. 0 is black, 1 is blue, 2 is green, 3 is cyan, 4 is red, 5 is magenta, 6 is yellow, 7 is white |
| *art* | a vector of integers, 0 ... The dimension of art must be equal to count of rows from p_matrix. 0 is unvisible, 1 is solid, 2 is dashed 1 to 1, 3 is dashed 1 to 2, 4 is dashed 1 to 3, 5 is dashed 2 to 1, 6 is dashed 2 to 2, 7 is dashed 2 to 3, 8 is dashed 3 to 1, 9 is dashed 3 to 2, and so on. |
| *thick* | a vector of integers, 0 ... , the thickness of the line The dimension of thick must be equal to count of rows from p_matrix. |

**Output:**

**Example:**

```
proc () = testline()
di=createdisplay(1, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/5
data=x~y
part=#(1, 1, 1, 1, 4)
i=1
while (i<20)
part=part| (#(1, 1, 1, 1, 4))
i=i+1
endo
; part contains now #(1, 1, 1, 4, 1, 1, 1, 4, ..., 1, 1, 1, 4)

setmaskp(data, part, 3, 8)
; make each fiveth point red circle

fiveth=5*(1:20)
; fiveth contains now #(5, 10, 15, 20, ..., 100)

tenth=10*(1:10) | (matrix(10)-1)
; tenth contains now #(10, 20, ..., 100, 0, .., 0)

setmaskl(data, (fiveth ~tenth)', #(4, 1), 1, #(2, 4))
; connect points 5, 10, ..., 100 with a solid red line,
; 2 pixels thick
; and points 10, 20, ..., 100 with a solid blue line,
; 4 pixels thick

show(di, 1, 1, data)
endp
testline()
```

**Result:** `you see it yourself.`

**Group:** Plot Manipulation
**See also:** setmaskl, setmaskt, show

## Function: setmaskp

**Description:** Use SETMASKP to control the color, the graphical representation (symbol) and the size of each datapoint of a datamatrix you want to plot using the SHOW command. SETMASKP has to preceed SHOW. That is, first you call SETMASKP to specify colour, graphical representation and size of the data you want to plot. Then you call SHOW to actually plot the data. You may assign the same colour to each datapoint (in which case the second argument of SETGOPT has to be a single integer) or you can assign different colours to different points (in which case the second argument of SETGOPT has to be a column vector of integers with the same number of rows as the datamatrix). See the definition of the inputs of SETGOPT for a complete description of which integer represents which colour. Similarly, you may represent each datapoint by the same graphical symbol (in which case the third argument of SETGOPT has to be a single integer) or you may represent different datapoints by a different graphical symbol (in which case the third argument of SETGOPT has too be a column vector of integers with the same number of rows as the datamatrix). See the definition of the inputs of SETGOPT for a complete description of which integer represents which grapical symbol. The fourth argument of SETGOPT allows you to control the size of the datapoints. You may not assign different sizes to different points. Rather, you specify an integer that determines the size of all datapoints. See the definition of the inputs of SETGOPT for a complete description of which integer represents which size.

**Usage:** *setmaskp(data, color_num, art_num, size)*

**Input:**

| | |
|---|---|
| *data* | (n x p) datamatrix |
| *color_num* | either a scalar integer (all points of the datamatrix get the color associated with that integer) or a (n x 1) vector of integers (each point gets the position associated with the integer in the corresponding row); 0 is black, 1 is blue, 2 is green, 3 is cyan, 4 is red, 5 is magenta, 6 is yellow, 7 is white |
| *art_num* | either a scalar integer (all points of the datamatrix get represented by the symbol associated with that integer) or a (n x 1) vector of integers (each point gets represented by the symbol associated with the integer in the corresponding row); 0 is empty, 1 is a point, 2 is a rectangle, 3 is a circle, 4 is a triangle, 5 is a X-symbol, 6 is a rhombus, 7 is a filled rectangle, 8 is a filled circle, 9 is a filled rhombus, 10 is a filled triangle, 11 is a cross, 12 is a star, 13 is a grid, 14 is a different cross |
| *size* | either a scalar integer (all points of the datamatrix get the size associated with that integer) or a (n x 1) vector of integers (each point gets the size associated with the integer in the corresponding row); size may be one of the following integers: 0 ... 16; the default size is 8. |

**Notes:** Technically, SETMASKP creates maskvectors that contain the numerical information to control the graphical display of the datapoints. This explains the name of the function.

**Example:**

```
di=createdisplay(1, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/5
data=x~y
setmaskp(data, 4, 3, 8)
show(di, 1, 1, data)
```

**Result:** all points of the datamatrix are coloured red (second argument equals 4), are shown as circles (third argument equals 3) and have the standard size (fourth argument equals 8).

**Example:**
```
di=createdisplay(1, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
data=x~y
color=4*matrix(50) | 5*matrix(50)
layout=3*matrix(25) | 4*matrix(25) | 5*matrix(25) | 6*matrix(25)
setmaskp(data, color, layout, 8)
show(di, 1, 1, data)
```

**Result:** you see the first 25 points as red circles,
second 25 point as red triangles,
third 25 points as  magenta crosses
and 25 last points as magenta rhombus.

**Group:** Plot Manipulation
**See also:** setmaskp, setmaskl, show

## Function: setmaskt

**Description:** SETMASKT is used to assign labels to each point of a datamatrix. If you have assigned labels with SETMASKT you can plot the points and their label using SHOW.
SETMASKT allows you to control the colour, the position (relative to the associated datapoint) and the fontsize of the label. The labels themselves have to be collected in a text vector and given to SETMASKT as an input. Colour, position and size are controlled by integers. See the definition of the inputs for a complete description of which integers are representing which colour, position or font. You can assign the same colour, position or fontsize to all labels of the datamatrix (in which case the respective inputs for colour, position or fontsize have to be given to SETMASKT as (scalar-)integers) or you may assign different colours, positions, or fontsizes to different labels (in which case the respective inputs have to be given to SETMASKT as vectors of integers with the same number of rows as the datamatrix).

**Usage:** *setmaskt(data, labels, colour, position, fontsize)*

**Input:**

| | |
|---|---|
| *data* | a (n x p) data matrix |
| *labels* | a text vector, with the same number of strings as the dataset has rows. |
| *colour* | either a scalar integer (all labels of the dataset get the colour associated with that integer) or a (n x 1) vector of integers (each label gets the colour associated with the integer in the corresponding row); 0 is black, 1 is blue, 2 is green, 3 is cyan, 4 is red, 5 is magenta, 6 is yellow, 7 is white |
| *position of the label relative to the corresponding datapoint* | either a scalar integer (all labels of the dataset get the position associated with that integer) or a (n x 1) vector of integers (each label gets the position associated with the integer in the corresponding row); -1 is for no label at all, 0 is for a centered label, 3 is for a label to the right of the point, 6 is for a label below the point, 9 is for a label to the left of the point, 12 is for a label above the point, |
| *fontsize of the labels* | either an integer (all labels get the same font size), or a (n x 1) vector of integers (each label gets the fontsize associated with the integer in the corresponding row) XploRe4 suppors the font sizes: 12 ... 16 |

**Notes:** Technically, SETMASKT creates maskvectors that contain the numerical information to control the graphical display of the labels. This explains the name of the function.

**Example:**

```
x=1:6
x=x~x
d=createdisplay(1, 1)
text="Point1"|"Point2"|"Point3"|"Point4"|"Point5"|"Point6"
setmaskt(x, text, 1, 3, 12)
show(d, 1, 1, x)
```

**Result:**

```
6 data points and their labels are shown. all labels are blue, to the
right of the datapoint and have fontsize 12
```

**Example:**

```
x=1:6
x=x~x
d=createdisplay(1, 1)
text="Right"|"Under"|"Left"|"Over"|"Center"|"No"
setmaskt(x, text, 1:6, 3|6|9|12|0|(-1), 10+2*x[,1])
show(d, 1, 1, x)
```

**Result:** you see 6 datapoints and their 6 labels. all 6 labels have a different layout.

**Group:** Plot Manipulation
**See also:** setbrushf, link, readevent

## Function: setmode

**Description:** setmode sets the mouse mode of a plot.
0=ZOOM
1=BRUSH
1=NOTHING

**Usage:** *setmode(di, l_num, c_num, mode)*

**Input:**

| | |
|---|---|
| *di* | Display |
| *l_num* | integer, 1 ... count of lines of di |
| *c_num* | integer, 1 ... count of columns of di |
| *mode* | 0 if you need the default mode (ZOOM and INDEX)<br>1 if you want to brush<br>2 if you want to define your own actions using readevent |

**Example:**

```
di=createdisplay(1, 2)
x=1:100
y1=x~(sin(x/20)+uniform(100, 1)/10)
y2=x~(cos(x/20)+uniform(100, 1)/10)
setmaskp(y1, 4, 3, 8)
setmaskp(y2, 4, 3, 8)
show(di, 1, 1, y1)
show(di, 1, 2, y2)
link(di, 1, 1, di, 1, 2)
setmode(di, 1, 1, 1)
```

**Result:**

```
after the mouse cursor enters the left plot, you see a
brush rectangle around the cursor. If you click a button
now, all the points that are in this rectangle are brushed.
Because of the link, the corresponding points in the right
plot will be black color too.
```

**Group:** Plot Manipulation
**See also:** axeson, axesoff

**Function:** **setsize**

**Description:** setsize sets the size for all the plots created later

**Usage:** *setsize(width, height)*

**Input:**

| | | |
|---|---|---|
| | *width* | integer |
| | *height* | integer |

**Output:**

**Example:**

```
setsize(250, 500)
di=createdisplay(2, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
show(di, 1, 1, x~y)
show(di, 2, 1, x~y)
```

**Result:** `A new display is created with chosen size.`

**Group:** Handling of time
**Subgroup:** Generating timepoints
**See also:** settimegran settimezero getnow getlocalnow

## Function: settime

**Description:** settime generates from a set of vectors a vector of time points. The parameter yr (year) is necessary, the parameters mo (month), dy (day of month), hr (hour), mn (minute), sc (second) are optional. The granulation of the measurement can be changed which the command setgran. Each value in v describes the distance to a fixed point in time (usually 1.1.1900 0:00:00.0). The zero time can be changed with the command setzero. Be careful that you not try to span too much time with a too small granulation !!

**Usage:** *t = settime (yr {, mo {, dy {, hr {, mn {, sc}}}}})*

**Input:**

| | |
|---|---|
| *yr* | n x 1 vector |
| *mo* | n x 1 vector |
| *dy* | n x 1 vector |
| *hr* | n x 1 vector |
| *mn* | n x 1 vector |
| *sc* | n x 1 vector |

**Output:** *t* n x 1 time vector

**Example:**

```
yr = 1970 + (1:10)
t  = settime (yr)
```

**Result:**

```
t contains now all dates from 1.1.1971 0:00:00.0 till
1.1.1980 0:00:00.0.
```

**Group:** Handling of time
**Subgroup:** Generating timepoints
**See also:** settime settimezero getnow getlocalnow

## Function: settimegran

**Description:** settimegran changes the granulation. Possible values for newgran are 1...7 (Seconds, minutes, hour, days, weeks, month, years)

**Usage:** *settimegran (t, newgran)*

**Input:**

| | |
|---|---|
| *t* | n x 1 time vector |
| *newgran* | scalar |

**Example:**

```
yr = 1970 + (1:10)
t  = settime (yr)
settimegran (t, 4)
```

**Result:**

```
t contains now all dates from 1.1.1971 0:00:00.0 till
1.1.1980 0:00:00.0 measured in days.
```

**Group:** Handling of time
**Subgroup:** Generating timepoints
**See also:** settime settimegran getnow getlocalnow

## Function: settimezero

**Description:** settimezero changes the fixed time point.

**Usage:** *settimezero (t, newgran)*

**Input:**

| | | |
|---|---|---|
| | *t* | n x 1 time vector |
| | *newgran* | scalar |

**Example:**

```
yr = 1970 + (1:10)
t  = settime (yr)
settimezero (t, 0)
```

**Result:**

```
t contains now all dates from 1.1.1971 0:00:00.0 till
1.1.1980 0:00:00.0 measured in days from 1.1.0 0:00:00.0
```

**Group:** Plot Manipulation
**See also:** setyaxis, show, setheadline

## Function: setxaxis

**Description:** setxaxis changes the layout of the horizontal axis.

**Usage:** *setxaxis(di, l_num, c_num, from, to{, origin, major, minor, m, k})*

| | | |
|---|---|---|
| **Input:** | *di* | Display |
| | *l_num* | integer, 1 ... count of lines of di |
| | *c_num* | integer, 1 ... count of columns of di |
| | *from* | the minimal value from the axis |
| | *to* | the maximal value from the axis |
| | *origin* | origin for tickmark |
| | *major* | major for tickmark |
| | *minor* | minor for tickmark |
| | *m* | m from the "m+k*x" as axis text |
| | *k* | k from the "m+k*x" as axis text |

**Example:**
```
di=createdisplay(1, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
show(di, 1, 1, x~y)
setxaxis(di, 1, 1, 0, 101, 0, 20, 5, 0, 1)
```

**Result:** `you see it yourself.`

**Group:** Plot Manipulation
**See also:** setxaxis, show, setheadline

**Function:** **setyaxis**

**Description:** setyaxis changes the layout of the vertical axis.

**Usage:** *setyaxis(di, l_num, c_num, from, to{, origin, major, minor, m, k})*

**Input:**

| | |
|---|---|
| *di* | Display |
| *l_num* | integer, 1 ... count of lines of di |
| *c_num* | integer, 1 ... count of columns of di |
| *from* | the minimal value from the axis |
| *to* | the maximal value from the axis |
| *origin* | origin for tickmark |
| *major* | major for tickmark |
| *minor* | minor for tickmark |
| *m* | m from the "m+k*y" as axis text |
| *k* | k from the "m+k*y" as axis text |

**Example:**

```
di=createdisplay(1, 1)
x=1:100
y=sin(x/20)+uniform(100, 1)/10
show(di, 1, 1, x~y)
setyaxis(di, 1, 1, -1.5, 1.5, 0, 0.5, 0.1, 0, 1)
```

**Result:** `you see it yourself.`

**Group:** Plot Manipulation
**See also:** adddata, createdisplay, deletedata, getdata, setmaskl, setmaskp, setmaskt

**Function:** **show**

**Description:** SHOW is used to put graphical objects into a display. There are many things that qualify as "graphical objects" and consequently SHOW has many uses. We will not describe all of them but a few illustrative examples will be given. For instance a datamatrix with (at least) 2 columns is a graphical object and SHOW may be used to obtain a scatterplot of the data in the first column versus the data in the second column. If you are impatient look at Example 3 below which illustrates the main uses of SHOW. Otherwise read ahead. Before you call SHOW you have to create a display using CREATEDISPLAY. SHOW is your tool to put the graphical object into this display. See Example 1 on how to put a 2-dimensional scatterplot of data into a one-window display. You may put several graphical objects into the same display-window. For instance, you may want to put a 2 dimensional datamatrix and a smooth of this data into the same display. Example 2 shows you how. Suppose you want to show several graphical objects simultaneously but not in the same display window: First, create a multi-window display and the use a separate SHOW command to fill each window with the desired graphical object. This is illustrated in Example3. SHOW is not limited to numerical graphical objetcs but may also be used to put strings in a display. See also Example 3. SHOW may not be used to change the appearance of your displays (like labeling axis, colouring and connecting points). This is done by other functions that are members of the function group "Plot Manipulation".

**Usage:** *show(di, l_num, c_num, data1, data2, ...)*

**Input:**

| | |
|---|---|
| *di* | Name of the display |
| *l_num* | integer, 1 ... count of rows of di |
| *c_num* | integer, 1 ... count of columns of di |
| *data1* | data matrix with min. 2 columns |

**Example:**
```
func("gennorm") ; load macro "gennorm"
sig=#(1,0.7)~#(0.7~1)  ; covariance matrix
z=gennorm(100,0|0,sig) ; create 100 observations of bivariate normal data
di1=createdisplay(1, 1)  ; creates a 1-window display
show(di1,1,1,z)          ; scatterplot of datamatrix z
```

**Example:**
```
func("gennorm") ; load macro "gennorm"
func("skerreg") ; load macro "skerreg"
sig=#(1,0.7)~#(0.7~1)  ; covariance matrix
z=gennorm(500,0|0,sig); create 100 observations of bivariate normal data
z=sort(z,1) ; sort data
{xs,mh}=skerreg(z[,1],z[,2],0.5)  ; kernel-smooth
di2=createdisplay(1,1)   ; create 1-window display
show(di2, 1, 1, z, xs~mh)  ; show datamatrix and smooth
```

**Example:**

```
func("skerreg") ; load macro "skerreg"
fubc("gls") ; load macro "gls"
di3=createdisplay(2, 2)  ; create a 4-window display
n=500 ; sample size
x=1:n ; generate explanatory variable
y=sin(x[,2]/100)+normal(500, 1)      ; generate dependent variable
b=gls(matrix(n)~x,y) ; OLS regression with intercept
yols=(matrix(n)~x)*b ; OLS  fit
{xs,mh}=skerreg(x,y,15)
; sorted explanatory variables (xs) and kernel fit (mh)
show(di3, 1, 1, x~y) ;scatterplot
show(di3, 1, 2, x~y,xs~mh) ; scatterplot and kernel smooth
; to display strings, put them in quotation marks
; to create a new line use stacking up operator "|"
; to combine strings with numerical elements use STRING function
show(di3, 2, 1, "OLS estimates"|string("intercept= %f", b[1,])|string("slope= %f", b[2,]))
show(di3, 2, 2, x~y, x~yols) ; scatterplot and OLS fit
```

**Group:** Mathematical Functions
**Subgroup:** Integration
**See also:**

## Function: simpsonint

**Description:** simpsonint computes the integral of a given realvalued function about a d-dimensional cube [a_1,b_1] x ... x [a_d,b_d]. The method of simpson is used.

**Usage:** *y = simpsonint (func, a, b, m)*

**Input:**

| | |
|---|---|
| *func* | text, name of the procedure where the function is defined |
| *a* | d vector, containing the begining points of the intervals. |
| *b* | d vector, containing the end points of the intervals. |
| *m* | d vector, 2 * m(i) is the number of subintervals in the i-th dimension. |

**Output:**

| | |
|---|---|
| *y* | scalar, value of the integral |

**Example:**

```
proc(y) = fun(x)
y=sum(x^3)
endp

a = #(0,0)
b = #(1,2)
m = #(3,6)
simpsonint("fun",a,b,m)
```

**Result:** `[1,] 4.500000`

**Group:** Mathematical Functions
**Subgroup:** Trigonometric Functions
**See also:** acos, acosh, asin, asinh, atan, atan2, atanh, cos, cosh, sinh, tan, tanh

## Function: sin

**Description:** sin returns the sine in radian for elements in an array.

**Usage:** *z = sin (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Example:**

```
x = #(pi/2, pi/4)~#(pi, 3/4*pi)
sin (x)
```

**Result:**

```
[1,]       1 1.2246e-16
[2,] 0.70711 0.70711
```

**Group:** Mathematical Functions
**Subgroup:** Trigonometric Functions
**See also:** acos, acosh, asin, asinh, atan, atan2, atanh, cos, cosh, sin, tan, tanh

## Function: sinh

**Description:** sinh returns the sinus hyperbolicus of the elements in an array.

**Usage:** *z = sinh (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Example:**
```
x = #(0, 2)~#(3, 4)
sinh (x)
```

**Result:**
```
[1,]      0 10.018
[2,] 3.6269 27.29
```

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** {}, comp, list

**Function:** size

**Description:** size gives the number of elements, containing in a list.

**Usage:** *y = size(x)*

**Input:** *x* list

**Output:** *y* integer

**Notes:** If the argument isn't a list, size will return 0.

**Example:**

```
x1=(1:4)'
x2=matrix(2,2)
y=list(x1,x2,(4:6)')
size(y)
```

**Result:**

```
[1,]     3
```

**Group:** Statistical Data Analysis
**Subgroup:** Nonparametric Methods
**See also:** isoreg sknn rmed l1line lowess

**Function:** **sker**

**Description:** sker computes a direct kernel estimate without binning from scatter plot data.

**Usage:** *z = sker (x, h, K, y {, xest})*

| **Input:** | | |
|---|---|---|
| | *x* | n x p x d1 x ... x dn array |
| | *h* | 1 x p x d1 x ... x dn array (global bandwidth) |
| | | n x p x d1 x ... x dn array (local bandwidth) |
| | *K* | text (kernel) |
| | *y* | n x m x d1 x ... x dn array |
| | *xest* | k x p x d1 x ... x dn array |

**Output:** *z* k x m x d1 x ... x dn array

**Notes:** 1. As inputs you have to specify the explanatory variable x, the dependent variable y and the smoothing parameter h. Optionally the parameter xest gives the points of estimation. If the parameter xest is not given the values of x will be used for xest.
z_i = 1/n sum_{k=1,...,n} K((x_k-z_i)/h_i)/h_i y_i

2. sker works only on sorted data x~y. Before applying sker the data must be previously sorted by x. This is done automatically by the macro "skerreg.xpl" or "skerdens.xpl". If the optional parameter xest is given, the data has to be sorted by the first column. If xest is not given, x will be used as xest !

3. The kernel that you specify must have a support on [-1,1], e.g. you cannot use the gaussian kernel directly. To use the gaussian kernel you an approximation like K(x) = 1/sqrt(2 pi) exp(-0.5 x2) I(abs(x)<M). If you transform your data by u = x/M and the kernel by K_M(x) = exp(-0.5/M2) exp(-0.5 u2) I(abs(u)<1) and the bandwidth you achieve a close approximation to the estimation with the gaussian kernel.

**Example:**
```
library("kernel")        ; loading the quartic kernel
x = 2.*uniform(100)-1
x = sort(x)
y = 5*x^2+0.5*normal(100)
d = createdisplay(1,1)
;
r = sker (x, 0.25, "qua", y)
f = sker (x, 0.25, "qua", matrix(100))
mh = r./f                ; nadaraya-watson estimator on x
show(d, 1, 1, x~y, x~mh)
; continuing using xest
xest = 2.*(0:100)/100-1
r = sker (x, 0.25, "qua", y, xest)
f = sker (x, 0.25, "qua", matrix(100), xest)
mh = r./f                ; nadaraya-watson estimator on xest
show(d, 1, 1, x~y, xest~mh)
```

**References:** haerdle, w. (1990). "applied nonparametric regression," cambridge university press.

**Group:** Statistical Data Analysis
**Subgroup:** Nonparametric Methods
**See also:** sker isoreg rmed l1line lowess

## Function: sknn

**Description:** sknn computes the k-nearest neighbour smooth regression from scatter plot data. As inputs you have to specify the explanatory variable x, the dependent variable y and the smoothing parameter k.

**Usage:** *yh = sknn (x, y, k)*

**Input:**

| | |
|---|---|
| *x* | n x 1 vector |
| *y* | n x 1 vector |
| *k* | scalar |

**Output:**

| | |
|---|---|
| *yh* | n x 1 vector |

**Notes:** sknn works only on sorted data x∼y. Before applying sknn the data must be previously sorted by x.

**Example:**

```
x = read ("geyser")
x = sort (x, #(1,2))
mh = x[,1]~sknn (x[,1],x[,2], 11)
d = createdisplay (1,1)
show (d, 1, 1, x, mh)
```

**Result:** `shows the geyser data and the sknn regression smoother`

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** index

**Function:** **sort**

**Description:** sort sorts the rows of a matrix. If column c1 is specified the matrix will be sorted with respect to column c1. That is, the rows of the matrix will be arranged in order that elements of column c1 are in ascending (descending) order.

**Usage:** *z = sort (x, c)*

**Input:**
*x* n x p x d1 x ...x dn array
*c* m x 1 vector

**Output:** *z* n x p x d1 x ...x dn array

**Notes:** If m>1 then the matrix will be first sorted by column c[1]. Then, will be sorted by column c[2] without changing the order given by c[1]. The process will continue until no further sort is being possible (without changing the previous order) or no more parameters are found.
If c[j] (j=1...m) is positive the sort will be in ascending order. to sort in descending order with respect to c[j] input as parameter -c[j].
To sort every column of matrix x separately use z=sort(x, 0).

**Example:**
```
x  = #(3, 1, 1, 3, 1)~#(4, 5, 4, 2, 4)~#(1, 5, 4, 0, 1)
sort (x', #(-1, 2))
```

**Result:**
```
[1,]     4     5     4     2     4
[2,]     3     1     1     3     1
[3,]     1     5     4     0     1
```

**Group:** Mathematical Functions
**Subgroup:** Common Functions
**See also:**

**Function:** **sqrt**

**Description:** sqrt computes the square root of the elements of an array.

**Usage:** *z = sqrt (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Notes:** NaN's are generated as a result of applying this operation to negative values.

**Example:** `sqrt (#(9, 25, 36)')`

**Result:**

```
[1,]     3     5     6
```

**Group:**
**See also:** atof substr

**Function:** **string**

**Description:** string converts a set of vectors through a format string in a string.

**Usage:** *z = string (fmt, v1 {, v2 {, v3 {, v4 {, v5}}}})*

| | | |
|---|---|---|
| **Input:** | *fmt* | formatstring |
| | *vi* | n x 1 vector |
| **Output:** | *z* | n x 1 vector |

**Notes:** The format has to follow the C-conventions. The following escape sequences produce the associated action on display devices capable of the action:

\a
Alert. Ring the bell.

\b
Backspace. Move the printing position to one character before the current position, unless the current position is the start of a line.

\f
Form feed. Move the printing position to the initial printing position of the next logical page.

\n
Newline. Move the printing position to the start of the next line.

\r
Carriage return. Move the printing position to the start of the current line.

\t
Horizontal tab. Move the printing position to the next implementation-defined horizontal tab position on the current line.

\v
Vertical tab. Move the printing position to the start of the next implementation-defined vertical tab position.

Each conversion specification is introduced by the character %. After the character %, the following appear in sequence:

An optional field, consisting of a decimal digit string followed by a $, specifying the next args to be converted. If this field is not provided, the args following the last args converted will be used.

Zero or more flags, which modify the meaning of the conversion specification.

An optional string of decimal digits to specify a minimum field width. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag (-), described below, has been given) to the field width.

An optional precision that gives the minimum number of digits to appear number of digits to appear after the decimal-point character for the e, E, and f conversions, the maximum number of significant digits for the g and G conversions.

The precision takes the form of a period (.) followed by a decimal digit string; a null digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.

A conversion character (see below) that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk ( *) instead of a digit string. In this case, an integer args supplies the field width or precision. The args that is actually converted is not fetched until the conversion letter is seen, so the args specifying field width or precision must appear before the args (if any) to be converted. If the precision argument is negative, it will be changed to zero. A negative field width argument is taken as a - flag, followed by a positive field width.

In format strings containing the *digits$ form of a conver- sion specification, a field width or precision may also be indicated by the sequence *digits$, giving the position in the argument list of an integer args containing the field width or precision.

When numbered argument specifications are used, specifying the Nth argument requires that all the leading arguments, from the first to the (N-1)th, be specified in the format string.

The flag characters and their meanings are:

-
The result of the conversion will be left-justified within the field. (It will be right-justified if this flag is not specified.)

+
The result of a signed conversion will always begin with a sign (+ or -). (It will begin with a sign only when a negative value is converted if this flag is not specified.)

space
If the first character of a signed conversion is not a sign, a space will be placed before the result. This means that if the space and + flags both appear, the space flag will be ignored.

#
The value is to be converted to an alternate form. For e, E, f, g, and G conver- sions, the result will always contain a decimal-point character, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For g and G conversions, trailing zeros will not be removed from the result as they normally are.

0
For e, E, f, g, and G conversions, leading zeros (following any indication of sign or base) are used to pad to the field width; no space padding is performed. If the 0 and flags both appear, the 0 flag will be ignored.

Each conversion character results in fetching zero or more args. The results are undefined if there are insufficient args for the format. If the format is exhausted while args remain, the excess args are ignored.

The conversion characters and their meanings are:

f
The double args is converted to decimal notation in the style [ - ]ddd.ddd, where the number of digits after the decimal-point character (see setlocale(3C)) is equal to the precision specification. If the precision is omitted from arg, six digits are output; if the precision is explicitly zero and the # flag is not specified, no decimal-point character appears. If a decimal-point character appears, at least 1 digit appears before it. The value is rounded to the appropriate number of digits.

e,E

The double args is converted to the style [ - ]d.ddde + dd, where there is one digit before the decimal-point character (which is non- zero if the argument is non-zero) and the number of digits after it is equal to the precision. When the precision is missing, six digits are produced; if the precision is zero and the # flag is not specified, no decimal-point character appears. The E conversion character will produce a number with E instead of e introducing the exponent. The exponent always contains at least two digits. The value is rounded to the appropriate number of digits.

g,G
The double args is printed in style f or e (or in style E in the case of a G conversion character), with the precision specifying the number of significant digits. If the precision is zero, it is taken as one. The style used depends on the value converted: style e (or E) will be used only if the exponent resulting from the conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional part of the result. A decimal-point character appears only if it is followed by a digit.

%
Print a %; no argument is converted.

If the character after the % or %digits$ sequence is not a valid conversion character, the results of the conversion are undefined.

If a floating-point value is the internal representation for infinity, the output is [ + ]Infinity, where Infinity is either Infinity or Inf, depending on the desired output string length. Printing of the sign follows the rules described above.

If a floating-point value is the internal representation for "not-a-number," the output is [+]NaN. Printing of the sign follows the rules described above.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result.

**Example:**

```
fmt = "Number  : %10.3f"
i   = 1:10
str = string (fmt, i)
str
```

**Result:**

```
[ 1,] Number  :      1.000
[ 2,] Number  :      2.000
[ 3,] Number  :      3.000
[ 4,] Number  :      4.000
[ 5,] Number  :      5.000
[ 6,] Number  :      6.000
[ 7,] Number  :      7.000
[ 8,] Number  :      8.000
[ 9,] Number  :      9.000
[10,] Number  :     10.000
```

**Group:** String Manipulation
**See also:** atof string

## Function: substr

**Description:** substr extracts a substring from a string.

**Usage:** *z = string (str, from, to)*

**Input:**

| | |
|---|---|
| *str* | n x 1 string vector |
| *from* | n x 1 vector |
| *to* | n x 1 vector |

**Output:**

| | |
|---|---|
| *z* | n x 1 string vector |

**Example:**

```
fmt = "Number  : %10.3f"
i   = 1:10
str = string (fmt, i)
str = substr (str, i, i+10)
str
```

**Result:**

```
[ 1,] Number  :
[ 2,] umber  :
[ 3,] mber  :
[ 4,] ber  :
[ 5,] er  :
[ 6,] r  :      6
[ 7,]   :      7.
[ 8,]  :      8.0
[ 9,] :      9.00
[10,]      10.000
```

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** cumprod, cumsum, mean, prod

**Function:** **sum**

**Description:** sum computes the sum of the elements of an array regarding a given dimension.

**Usage:** $z = sum\ (x\ \{,j\})$

**Input:** $x$ d1 x d2 x ... x dn array
$j$ integer (number of dimension, default = 1)

**Output:** $z$ d1 x d2 x .. x d(j-1) x 1 x d(j+1) x .. x dn array

**Example:**
```
x = #(1, 2)~#(3, 4)~#(5, 6)
sum (x,2)
```

**Result:**
```
[1,]     9
[2,]    12
```

**Group:** Linear Algebra
**See also:** eigsm

## Function: svd

**Description:** computes the singular value decomposition of a n x p matrix x (n >= p). The singular value decomposition finds matrices u, l, v such that x = u*l*v' u, v are orthogonal matrices l is a diagonal matrix

**Usage:** *y = svd(x)*

**Input:** *x* n x p matrix

**Output:**
*y.u* n x p matrix, the left matrix u
*y.l* p vector, elements of the diagonale of matrix l
*y.v* p x p matrix, the right matrix v

**Example:**

```
x = #(1:5)~#(6:10)~#(11:15)
y = svd(x)
r = x - y.u * (y.l .* y.v')
r
```

**Result:**

```
[1,]  8.1255e-09 -1.6251e-08  8.1255e-09
[2,]  4.2328e-08 -8.4655e-08  4.2328e-08
[3,] -4.4216e-08  8.8433e-08 -4.4216e-08
[4,] -7.1053e-08  1.421e-07 -7.1052e-08
[5,]  6.4816e-08 -1.2963e-07  6.4816e-08
```

**Group:** Flow Control
**See also:** case break default endsw

## Function: switch

**Description:** switch allows the selection of one or more alternatives of many. Each alternative is introduced by a case statement. Similar to if-endif it controls, whether the following block is processed or not. The keyword break serves as end marker of case and leaves the switch block at the position of endsw. When break is omitted, the next consecutive case is processed. If the program's counter comes to default, the following block is executed in any case.

**Usage:** *switch case (cond) break default endsw*

**Notes:** Even if the condition is a multidimensional array only the first element is checked for TRUE (!=0) or FALSE (==0)

**Example:**

```
proc (s) = sign (x)
switch
case (x<0)  y = -1  break
case (x>0)  y =  1  break
default     y =  0  break   ; this break has no effect
endsw
endp
```

**Group:** Mathematical Functions
**Subgroup:** Trigonometric Functions
**See also:** acos, acosh, asin, asinh, atan, atan2, atanh, cos, cosh, sin, sinh, tanh

## Function: tan

**Description:** tan returns the tangent in radians of the elements of an array.

**Usage:** *z = tan (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Example:**
```
x = #(pi/3, pi/4)~#(pi, 3/4*pi)
tan (x)
```

**Result:**
```
[1,] 1.7321 -1.2246e-16
[2,]      1    -1
```

**Group:** Mathematical Functions
**Subgroup:** Trigonometric Functions
**See also:** acos, acosh, asin, asinh, atan, atan2, atanh, cos, cosh, sin, sinh, tan

## Function: tanh

**Description:** tanh returns the hyperbolic tangent of the elemnts of an array.

**Usage:** *z = tanh (x)*

**Input:** *x* d1 x d2 x ... x dn array

**Output:** *z* d1 x d2 x ... x dn array

**Example:**
```
x = #(1, 2)~#(3, 4)
tanh (x)
```

**Result:**
```
[1,] 0.76159 0.99505
[2,] 0.96403 0.99933
```

**Group:** Linear Algebra
**See also:** ', inv

## Function: trans

**Description:** trans transposes matrices. This function is equal to the operator '

**Usage:** *y = trans (x)*

**Input:** *x* array of matrices

**Output:** *y* array of matrices

**Example:**
```
x = #(1,2)~#(3,4)~#(5,6)
trans (x)
```

**Result:**
```
[1,]     1     2
[2,]     3     4
[3,]     5     6
```

**Group:** Random Variables
**See also:** normal, randomize

## Function: uniform

**Description:** uniform generates arrays up to eight dimensions of pseudo random variables with a standard uniform distribution.

**Usage:** *z = uniform (d1 {, .., dn})*

**Input:** *d1, .., dn* scalar (number of elements in dimension 1 .. n)

**Output:** *z* d1 x d2 x ... x dn matrix containing pseudo random variables with a standard uniform distribution

**Example:**
```
randomize(0)
uniform (2, 3)
```

**Result:**
```
[1,] 1.1156e-07 0.12478 0.0035468
[2,] 0.0030269 0.42072 0.23159
```

**Group:** Flow Control
**See also:** do, endo, while

## Function: until

**Description:** until ends a do loop until the first element of the argument is true. do has to be followed by an until.

**Usage:** *do <xplore commands> until (cond)*

**Input:** *cond* n x 1 vector of condition

**Notes:** Even if the condition is a multidimensional array only the first element is checked for TRUE (!=0) or FALSE (==0)

**Example:**

```
proc(j)=factorial(x)
if (x>1)
j=1
do
j=j*x
x=x-1
until (x<2)
else
j=x
endif
endp
```

**Group:** Statistical Data Analysis
**Subgroup:** Basic Statistics
**See also:** cov, mean

**Function:** **var**

**Description:** var computes the variance of the elements of an array regarding a given dimension.

**Usage:** *z = var (x {j})*

**Input:** *x* d1 x d2 x ... x dn array
*j* integer (number of dimension, default = 1)

**Output:** *z* d1 x d2 x .. x d(j-1) x 1 x d(j+1) x .. x dn array

**Example:**

```
randomize(0)
x = normal(2, 4)
var(x,2)
```

**Result:**

```
[1,] 2.0185
[2,] 1.0476
```

**Group:** Matrix Manipulation
**Subgroup:** Sorting and Counting
**See also:** #, reshape

## Function: vec

**Description:** vec reshapes all given arguments into a vectors and concatenates them into a single vector which is returned.

**Usage:** *y = vec (x1, x2, ...)*

**Input:** *x1, ...* numerical arrays

**Output:** *y* vector

**Notes:** vec and # are identical.

**Example:** `vec(2,matrix(2,2),8)`

**Result:**

```
[1,]      2
[2,]      1
[3,]      1
[4,]      1
[5,]      1
[6,]      8
```

**Group:** Flow Control
**See also:** break, error, exit, endp, proc

## Function: warning

**Description:** warning enables the user to send warning messages out of macros. There are displayed when the whole program finishes.

**Usage:** *warning(x, m)*

**Input:**

| | |
|---|---|
| *x* | condition |
| *m* | string, message |

**Example:**

```
x = 0
warning (x == 0, "x is equal to zero")
```

**Result:**

```
XploRe_Warning_box containing:
x is equal to zero
```

**Group:** Flow Control
**See also:** do, endo, until

## Function: while

**Description:** while begins a loop if the first element of the argument is true. while has to be followed by an endo.

**Usage:** *while (cond) <xplore commands> endo*

**Input:** *cond* n x 1 vector of condition

**Notes:** Even if the condition is a multidimensional array only the first element is checked for TRUE (!=0) or FALSE (==0)

**Example:**

```
proc(j)=factorial(x)
if (x>1)
j=1
while (x>=2)
j=j*x
x=x-1
endo
else
j=x
endif
endp
```

**Group:** Reading and Writing
**See also:** read, readm, savemac

**Function:** **write**

**Description:** write writes an array into a file.

**Usage:** *write (x, file {, f})*

**Input:**

| | |
|---|---|
| *x* | multidimensional array |
| *file* | string with the name of the file |
| *f* | string, format |

**Notes:** If no format is specified, the ascii-format is used. Use formatstring "x" to save the data in the XploRe-format. Otherwise, the format has to follow the C-conventions. The following escape sequences produce the associated action on display devices capable of the action:

\b
Backspace. Move the printing position to one character before the current position, unless the current position is the start of a line.

\f
Form feed. Move the printing position to the initial printing position of the next logical page.

\n
Newline. Move the printing position to the start of the next line.

\r
Carriage return. Move the printing position to the start of the current line.

\t
Horizontal tab. Move the printing position to the next implementation-defined horizontal tab position on the current line.

\v
Vertical tab. Move the printing position to the start of the next implementation-defined vertical tab position.

Each conversion specification is introduced by the character %. After the character %, the following appear in sequence:

An optional field, consisting of a decimal digit string followed by a $, specifying the next args to be converted. If this field is not provided, the args following the last args converted will be used.

Zero or more flags, which modify the meaning of the conversion specification.

An optional string of decimal digits to specify a minimum field width. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag (-), described below, has been given) to the field width.

An optional precision that gives the minimum number of digits to appear number of digits to appear after the decimal-point character for the e, E, and f conversions, the maximum number of significant digits for the g and G conversions.

The precision takes the form of a period (.) followed by a decimal digit string; a null digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.

A conversion character (see below) that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk ( *) instead of a digit string. In this case, an integer args supplies the field width or precision. The args that is actually converted is not fetched until the conversion letter is seen, so the args specifying field width or precision must appear before the args (if any) to be converted. If the precision argument is negative, it will be changed to zero. A negative field width argument is taken as a - flag, followed by a positive field width.

In format strings containing the *digits$ form of a conver- sion specification, a field width or precision may also be indicated by the sequence *digits$, giving the position in the argument list of an integer args containing the field width or precision.

When numbered argument specifications are used, specifying the Nth argument requires that all the leading arguments, from the first to the (N-1)th, be specified in the format string.

The flag characters and their meanings are:

-
The result of the conversion will be left-justified within the field. (It will be right-justified if this flag is not specified.)

+
The result of a signed conversion will always begin with a sign (+ or -). (It will begin with a sign only when a negative value is converted if this flag is not specified.)

space
If the first character of a signed conversion is not a sign, a space will be placed before the result. This means that if the space and + flags both appear, the space flag will be ignored.

#
The value is to be converted to an alternate form. For e, E, f, g, and G conver- sions, the result will always contain a decimal-point character, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For g and G conversions, trailing zeros will not be removed from the result as they normally are.

0
For e, E, f, g, and G conversions, leading zeros (following any indication of sign or base) are used to pad to the field width; no space padding is performed. If the 0 and flags both appear, the 0 flag will be ignored.

Each conversion character results in fetching zero or more args. The results are undefined if there are insufficient args for the format. If the format is exhausted while args remain, the excess args are ignored.

The conversion characters and their meanings are:

f
The double args is converted to decimal notation in the style [ - ]ddd.ddd, where the number of digits after the decimal-point character (see setlocale(3C)) is equal to the precision specification. If the precision is omitted from arg, six digits are output; if the precision is explicitly zero and the # flag is not specified, no decimal-point character appears. If a decimal-point character appears, at least 1 digit appears before it. The value is rounded to the appropriate number of digits.

e,E

The double args is converted to the style [ - ]d.ddde + dd, where there is one digit before the decimal-point character (which is non- zero if the argument is non-zero) and the number of digits after it is equal to the precision. When the precision is missing, six digits are produced; if the precision is zero and the # flag is not specified, no decimal-point character appears. The E conversion character will produce a number with E instead of e introducing the exponent. The exponent always contains at least two digits. The value is rounded to the appropriate number of digits.

g,G
The double args is printed in style f or e (or in style E in the case of a G conversion character), with the precision specifying the number of significant digits. If the precision is zero, it is taken as one. The style used depends on the value converted: style e (or E) will be used only if the exponent resulting from the conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional part of the result. A decimal-point character appears only if it is followed by a digit.

%
Print a %; no argument is converted.

If the character after the % or %digits$ sequence is not a valid conversion character, the results of the conversion are undefined.

If a floating-point value is the internal representation for infinity, the output is [ + ]Infinity, where Infinity is either Infinity or Inf, depending on the desired output string length. Printing of the sign follows the rules described above.

If a floating-point value is the internal representation for "not-a-number," the output is [+]NaN. Printing of the sign follows the rules described above.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result.

**Example:**

```
write(reshape(1:15,#(3,5)),"my_data.dat","x")
read("my_data.dat")
```

**Result:**

```
[1,]     1     4     7    10    13
[2,]     2     5     8    11    14
[3,]     3     6     9    12    15
```

**Group:** Time and Date functions
**Subgroup:** Time as string output
**See also:** month dayofmonth dayofweek hour minute second

## Function: year

**Description:** returns the year as a string

**Usage:** *ts = year(t)*

**Input:** *t* n x 1 time vector

**Output:** *ts* n x 1 string array

**Example:**

```
yr = 1970 + (1:10)
t  = settime(yr)
ts = year(t)
ts
```

**Result:**

```
[ 1,] 1971
[ 2,] 1972
[ 3,] 1973
[ 4,] 1974
[ 5,] 1975
[ 6,] 1976
[ 7,] 1977
[ 8,] 1978
[ 9,] 1979
[10,] 1980
```