matrix.help - Function

Package	**utility**
Module	**mathematics**
Tool	**matrix**

print a summary of matrix functions

Synopsis
help()

---

matrix.test - Function

Package	**utility**
Module	**mathematics**
Tool	**matrix**

a test of most matrix functions

Synopsis
test()

1.5.30 lsfit - Tool

Package	**utility**
Module	**mathematics**

Least-squares fit by matrix inversion
include "lsfit.g"

Constructors

Functions

accumulate	accumulate an equation
help	print summary of lsfit functions
init	initialise before accumulation
inityonly	initialise the vector only

print	print internal matrices
solve	solves for the unknowns (x)
status	print lsfit internal status
test	test of the main lsfit functions

Description

The functions in this tool have not been tested as rigorously as other parts of AIPS++. This tool can be considered "alpha" code.
`lsfit` The main function of this tool is simultaneous solution of a set of linear equations by matrix inversion. If the matrix is rectangular, i.e. if there are more equations than unknowns, this amounts to a least-squares fit.

$$\vec{y} = A\,\vec{x} \tag{1.22}$$
$$A^{*T}\,\vec{y} = (A^{*T}A)\,\vec{x} \tag{1.23}$$

in which (*) indicates complex conjugation. The latter expression represents a set of **normal equations**, which can be solved by matrix inversion:

$$\vec{x} = (A^{*T}A)^{-1}\,(A^{*T}\,\vec{y}) \tag{1.24}$$

Various matrix inversion functions are supported: Gauss-Jordan, Choleski and Singular Value Decomposition (SVD). The latter automatically takes care of singular matrices, i.e. those cases where one is trying to solve for unknowns without supplying sufficient information. The default inversion function is Choleski decomposition, because of its efficiency.
The procedure is as follows (see also the example below): It is assumed that the number of unknowns (i.e. the length of the vector $\vec{x}$) is known, but that the equations $\vec{y} = A\,\vec{x}$ will become available one by one. First, an $n \times n$ matrix $(A^{*T}A)$ and a vector $(A^{*T}\,\vec{y})$ with length n are initialised with zeroes. Then, as each equation i becomes available, its coefficients a_{ij} and 'driving' value y_i are accumulated in the matrix and the vector. The system can be solved by matrix inversion as soon as at least n equations have been accumulated.
Since the solution does not destroy the accumulation matrix and vector, one may continue to accumulate more equations after a solve. In this way, one may get intermediate results.
If one needs more than one solution for different driving vectors $\vec{y}$, but the same matrix coefficients a_{ij}, one may use the same inverted matrix each time, and only re-accumulate the vector $(A^{*T}\,\vec{y})$. This saves time if the n is large.
The lsfit tool is a place-holder, until this functionality will be provided by a fitter DO. In the meantime, it is a good example of the power of prototyping in Glish, to find out what functionality and interface is really needed.

Example

NB: In view of the loop, it is recommended to make a .g script to try this example.

```
include "lsfit.g"
nuk := 5;                          # nr of unknowns
print 'xxin=',xxin := [1:nuk];    # simulated input values
lsf.init(nuk);                    # initialise matrix and vector
print lsf.status();                # optional
for (i in [1:(nuk-1)]) {
    for (j in [(i+1):nuk]) {
        aa := rep(0,nuk);          # coefficient vector
        aa[i] := 1;
        aa[j] := 1;
        y := xxin[i]*aa[i] + xxin[j]*aa[j]; # driving value
        lsf.accumulate(aa,y);      # accumulate equation
    }
}
print 'xxout=',xxout := lsf.solve(); # estimated values
```

lsfit.init - Function

Package	**utility**
Module	**mathematics**
Tool	**lsfit**

initialise before accumulation

Synopsis
init(nuk, keep, mode)

Arguments

nuk	nr of unknowns (Input) Allowed:int
keep	keep the coefficent matrix (Input) Allowed:bool Default: T
mode	inversion mode (Input) Allowed:string Default: 'chol'

Returns
Bool

Description
Initialises and zero $n \times n$ matrix, and a vector of length n, in which the information from equations $y_i = \sum_j a_{ij} x_j$ can be accumulated. If keep=T, the coefficients of the accumulated constraint equations are kept (see function solve). The various possible inversion modes are Choleski ('chol'), Gauss-Jordan ('gauss') and Singular Value Decomposition ('svd').

---

lsfit.accumulate - Function

Package	**utility**
Module	**mathematics**
Tool	**lsfit**

accumulate an equation

Synopsis
accumulate(aa, y, wgt, constreq, show)

Arguments

aa	coefficient vector (Input)
y	driving value (Input)
wgt	weight (Input) Default: 1.0
constreq	is constraint equation (Input) Allowed:bool Default: F
show	print one-line progress message (Input) Allowed:bool Default: F

Returns
Bool

Description
Accumulates the information from an equation $y_i = \sum_j a_{ij} x_j$ into the

internal matrix and vector (see example above). The weight can be used to reduce the influence of less reliable data on the solution. A constraint equation is one that is not driven by data, but provides a-priori information, e.g. to make the system soluble. This information allows the system to behave more intelligently in some cases.

---

lsfit.solve - Function

Package	**utility**
Module	**mathematics**
Tool	**lsfit**

solves for the unknowns (x)

Synopsis
solve(yy)

Arguments

yy	vector of new driving values (Input) Allowed:double Default: F

Returns
solution vector

Description
Returns a solution vector $\vec{x}$ of values for the unknowns. The accumulation matrix and vector are not affected, so accumulation may continue after a solve. In general, the solution is for the y-values given to .accumulate(). But it is also possible to solve the same system of equation for a new vector (yy), without doing a new inversion. Obviously, yy must be of the same length as the original one, and the same weights are used. For this option to work, the switch 'keep' set in init() must be T (default). The new vector yy replaces the internal vector.

Example

```
include "lsfit.g"
nuk := 5;                           # nr of unknowns
print 'xxin=',xxin := [1:nuk];      # simulated input values
lsf.init(nuk);                      # initialise matrix and vector
print lsf.status();                 # optional
for (i in [1:(nuk-1)]) {
    for (j in [(i+1):nuk]) {
        aa := rep(0,nuk);           # coefficient vector
        aa[i] := 1;
        aa[j] := 1;
        yy1[i] := xxin[i]*aa[i] + xxin[j]*aa[j]; # driving value
        lsf.accumulate(aa,yy1[i]);      # accumulate equation
    }
}
xx1 := lsf.solve( );   # solve for input vector yy1
xx2 := lsf.solve(yy2); # solve for other vector yy2
xx3 := lsf.solve(yy3); # solve for other vector yy3
xx4 := lsf.solve(yy4); # solve for other vector yy4
```

lsfit.inityonly - Function

Package	**utility**
Module	**mathematics**
Tool	**lsfit**

initialise the vector only

Synopsis

inityonly()

Description

For large numbers of equations, it may be too expensive to keep the coefficients of the input equations. In this case it is still possible to solve for a new yy-vector without a new inversion.

Example

```
lsf.init(nuk);                     # initialise matrix and vector
for (i in [1:(nuk-1)]) {
    for (j in [(i+1):nuk]) {
```

```
        aa := rep(0,nuk);         # coefficient vector
        aa[i] := 1;
        aa[j] := 1;
        lsf.accumulate(aa,yy1[i],wgt[i]);     # accumulate equation
    }
}
xx1 := lsf.solve(); # solve for input vector yy1

lsf.inityonly();                   # initialise matrix and vector
for (i in [1:(nuk-1)]) {
    for (j in [(i+1):nuk]) {
        aa := rep(0,nuk);         # coefficient vector
        aa[i] := 1;
        aa[j] := 1;
        lsf.accumulate(aa,yy2[i]);     # accumulate equation
    }
}
xx2 := lsf.solve(); # solve for vector yy2
```

lsfit.status - Function

Package	**utility**
Module	**mathematics**
Tool	**lsfit**

print lsfit internal status

Synopsis
status(name)

Arguments

name	Optional: header string (Input) Default: ' '

Returns
string

Description
Returns a string that describes the internal status of the lsf tool, which can be printed.

lsfit.print - Function

Package	**utility**
Module	**mathematics**
Tool	**lsfit**

print internal matrices

Synopsis
print(name)

Arguments

name	name of matrix to be printed (Input) Allowed:string

Description

lsf.print('ata') prints the accumulation matrix.
lsf.print('inv') prints its inverse.
lsf.print('aty') prints the accumulation vector.
lsf.print('status') prints the internal status.
lsf.print('self') prints the internal self record.

---

lsfit.help - Function

Package	**utility**
Module	**mathematics**
Tool	**lsfit**

print summary of lsfit functions

Synopsis
help()

---

lsfit.test - Function

Package	**utility**
Module	**mathematics**
Tool	**lsfit**