14 Modelling of data

Functions:

fit: fit data to a straight line using least squares
lfit: linear least squares fit, general using normal equations
covsrt: covariance matrix, sort, used by lfit
svdfit: linear least squares fit, general, singular value decomposition
svdvar: variances from singular value decomposition
fpoly: fit a polynomial, using lfit or svdfit
fleg: fit a Legendre polynomial, using lfit or svdfit
mrqmin: nonlinear least squares fit Marquardt's method
mrqcof: nonlinear least squares fit, used by mrqmin
fgauss: fit a sum of Gaussians, using mrqmin
medfit: fit data to a straight line robustly, least absolute deviation

fit

Description:

This function fits data to a straight line using the method of least squares.

fit(x, y, sig, mwt) $\longrightarrow$ **[a, b, siga, sigb, chisq, q]**

Where:

x is a real list representing the first coordinates of a set of data points,
y is a real list representing the second coordinates of a set of data points,
sig is a real list representing the standard deviations of the data points,
mwt is an integer such that if its value is 0 the standard deviations are assumed to be unavailable, q is returned as 1d0 and the normalization of $chisq$ is to unit standard deviation on all points,
a is a floating point number representing the constant term in the regression line $y = a + bx$,
b is a floating point number representing the coefficient of x in the regression line,
siga is a floating point number representing the probable uncertainty in the coefficient a,
sigb is a floating point number representing the probable uncertainty in the coefficient b,
chisq is a floating point number being the chi-squared statistic,
q is a floating point number representing a goodness-of-fit measure.

Example:

Choose points on the line $y = 3x - 5$.

```
-> x : [-1,0,1,2,3,4,5,6,7,8,9];
-> y : 3*x - const(5,length(x))?
[-8, -5, -2, 1, 4, 7, 10, 13, 16, 19, 22]
-> sig : const(0.5,length(x))?
[0.5000000, 0.5000000, 0.5000000, 0.5000000, 0.5000000, 0.5000000,
 0.5000000, 0.5000000, 0.5000000, 0.5000000, 0.5000000]
-> fit(x,y,sig,0)?
[-5.0000000, 3.0000000, 0.0000000, 0.0000000, 0.0000000, 1.0000000]
-> Should get perfect fit! #
-> fit(x,y,sig,1)?
[-5.0000000, 3.0000000, 0.2430862, 0.0476731, 0.0000000, 1.0000000]
```

lfit

Description:

This function preforms a linear least squares fit using general normal equations.

lfit(x, y, sig, mfit, lista, funcs) $\longrightarrow$ **[a, covar, chisq]**

Where:

x is a real list representing the first coordinates of a set of data points,
y is a real list representing the second coordinates of a set of data points,
sig is a real list representing the standard deviations of the data points,
linear regression equation

$$y = \sum_{i=1}^{ma} A_i afunc_i(x)$$

mfit is the number of coefficients to be determined,
lista is an integer list of length ma such that the first $mfit$ elements represent the variables to be fitted and such that the rest are held constant,
funcs is a list of the names of ma Senac functions of one real variable returning the values of the basis functions,
a is a floating point list representing the parameters which have been computed,
covar is a floating point matrix representing the covariances,
chisq is a floating point number representing the value of chi-square.

Example:

This example takes a sample set of points on the cubic $f(x) = 1 + 2x + 3x^2 + 4x^3$ and then makes a call to `lfit`. Notice that `lfit` does not, in this case, get the same coefficients as $f(x)$ above has, however they give a polynomial which evaluates very nearly the same as $f(x)$ does at the points chosen.

```
-> f(x) := 1 + 2*x + 3*x^2 + 4*x^3;
-> Suitable basis set of polynomials is: #
-> f1(x) := 1;
-> f2(x) := x;
-> f3(x) := x^2;
-> f4(x) := x^3;
-> funcs : [f1,f2,f3,f4];
-> mfit : 3;
-> lista : [1,2,3,4];
-> x : [1,2,3,4];
-> y : map(f,x)?
[10, 49, 142, 313]
-> sig : const(0.1,length(x))?
[0.1000000, 0.1000000, 0.1000000, 0.1000000]
-> res : lfit(x,y,sig,mfit,lista,funcs)?
[[43.0000000, -64.8000000, 33.0000000, 0.0000000],
| 0.0775000   -0.0675000  0.0125000   0.0000000 |
|                                               |
| -0.0675000  0.0645000   -0.0125000  0.0000000 |
|                                               |, 2879.9999142]
| 0.0125000   -0.0125000  0.0025000   0.0000000 |
|                                               |
| 0.0000000   0.0000000   0.0000000   0.0000000 |
-> a : first(res);
-> solution(x) := first(a)*f1(x) + proj(a,2)*f2(x) + proj(a,3)*f3(x)
     + last(a)*f4(x);
-> Compare this list with y above: #
-> map(solution,x)?
[11.2000000, 45.4000000, 145.6000000, 311.8000000]
```

covsrt

Description:

This function sorts a covariance matrix - it is used internally by $lfit$.

covsrt(covar, ma, lista) ⟶ covar

Where:

covar is a real covariance $mfit$ by $mfit$ matrix for the subset of fitted parameters on input and on output a repacked matrix to the true order of the parameters, ma by ma, and with elements associated with fixed parameters being zero,
ma is the total number of parameters,
lista is an integer list of length $mfit$ representing the ordering of the fitted parameters.

Example:

This example tests `covsrt` with respect to the given matrix `covar`.

```
-> covar : matrix([1,2,3,4,5],
                  [2,3,4,5,6],
                  [3,4,5,6,7],
                  [4,5,6,7,8],
                  [5,6,7,8,9]);
-> lista : [2,4,6,8,10];
-> covsrt(covar,10,lista)?
| 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 |
|                                                            |
| 0.00  1.00  0.00  2.00  0.00  3.00  0.00  4.00  0.00  5.00 |
|                                                            |
| 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 |
|                                                            |
| 0.00  2.00  0.00  3.00  0.00  4.00  0.00  5.00  0.00  6.00 |
|                                                            |
| 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 |
|                                                            |
| 0.00  3.00  0.00  4.00  0.00  5.00  0.00  6.00  0.00  7.00 |
|                                                            |
| 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 |
|                                                            |
| 0.00  4.00  0.00  5.00  0.00  6.00  0.00  7.00  0.00  8.00 |
|                                                            |
| 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 |
|                                                            |
| 0.00  5.00  0.00  6.00  0.00  7.00  0.00  8.00  0.00  9.00 |
```

svdfit

Description:

This function performs a linear least squares fit using a singular value decomposition.

svdfit(x, y, sig, a, funcs) ⟶ [a, u, v, w, chisq]

Where:

x is a real list representing the first coordinates of a set of data points,
y is a real list representing the second coordinates of a set of data points,
sig is a real list representing the standard deviations of the data points,
funcs is the name of a Senac function of one real variable that returns a list of length ma representing the basis functions for the fit,
a is a floating point list representing on input approximate values for the fitting parameters and on output the computed values of the fit parameters,
u is a floating point matrix being part of the singular value decomposition,
v is a floating point matrix being part of the singular value decomposition,
w is a floating point list being part of the singular value decomposition,
chisq is a floating point number beng the chi-squared statistic.

Pre-variables:

svdfit_tol [1.0d-5] a floating point number used to cast out singular values which are small.

Example:

Here we take the polynomial $f(x) = 1 + 2x + 3x^2 + 4x^3$, as for `lfit`, and perform the same test as for that function. Notice that `svdfit` *does* get the same coefficients as the original $f(x)$.

```
-> f(x) := 1 + 2*x + 3*x^2 + 4*x^3;
-> Suitable basis set of polynomials is: #
-> f1(x) := 1;
-> f2(x) := x;
-> f3(x) := x^2;
-> f4(x) := x^3;
-> funcs : [f1,f2,f3,f4];
-> a : [1,2,3,4];
-> x : [1,2,3,4];
-> y : map(f,x)?
[10, 49, 142, 313]
-> sig : const(0.1,length(x))?
[0.1000000, 0.1000000, 0.1000000, 0.1000000]
-> svdfit(x,y,sig,a,funcs)?
[[1.0000000, 2.0000000, 3.0000000, 4.0000000],
| -0.0180622  0.3768995   -0.8488797  0.3701673  |
|                                                |
| -0.1226241  0.6613389   -0.0262427  -0.7395306 |
|                                                |,
| -0.3936582  0.5545786   0.4917251   0.5437667  |
|                                                |
| -0.9108624  -0.3361848  -0.1921487  -0.1427877 |
| -0.0199190  0.3438083   -0.7880494  0.5102718  |
|                                                |
| -0.0701233  0.5522717   -0.2667030  -0.7867327 |
|                                                |, [725.5414050, 36.5503761,
| -0.2567087  0.7207874   0.5439866   0.3444484  |
|                                                |
| -0.9637357  -0.2392853  -0.1092072  -0.0450525 |
7.3034249, 0.6195846]]
-> Compare with the coefficients of f above #
```

svdvar

Description:

This function computes variances from a singular value decomposition.

svdvar(v, ma, w) ⟶ cvm

Where:

v is a real matrix as returned by *svdfit*,
ma is an integer representing the total number of fit parameters,
w is a real matrix as returned by *sbdfit*,
cvm is a floating point matrix being the covariance matrix of the fit.

Example:

```
-> w : [0.0,1.0,2.0,3.0,4.0,5.0];
-> v : matrix([1.0, 1.0, 1.0, 1.0, 1.0, 1.0],
              [2.0, 2.0, 2.0, 2.0, 2.0, 2.0],
              [3.0, 3.0, 3.0, 3.0, 3.0, 3.0],
              [4.0, 4.0, 4.0, 4.0, 4.0, 4.0],
              [5.0, 5.0, 5.0, 5.0, 5.0, 5.0],
              [6.0, 6.0, 6.0, 6.0, 6.0, 6.0]);
-> svdvar(v,3,w)?
| 1.2500000  2.5000000  3.7500000  |
|                                  |
| 2.5000000  5.0000000  7.5000000  |
|                                  |
| 3.7500000  7.5000000  11.2500000 |
```

mrqmin

Description:

This function produces a nonlinear least squares fit using Marquardt's method.

mrqmin(x, y, sig, a, lista, mfit, funcs, funcsd, alamda) $\longrightarrow$ [covar, alpha, chisq, alamda]

Where:

x is a real list representing the first coordinates of a set of data points,
y is a real list representing the second coordinates of a set of data points,
sig is a real list representing the standard deviations of the data points,
a is a real list of length ma on input being an initial guess for the fitting parameters and on output current best fit for the parameters,
lista is an integer list numbering the elements of a such that the first $mfit$ elements are the ones that are adjusted with the remaining elements held fixed at their input values,
mfit is an integer representing the number of parameters to be fitted,
funcs is the name of a Senac function of $1+ma$ real variables x and the parameters a returning the value of the fitting function $yfit(x, a_1, ..., a_{ma})$,
funcsd is a list of length ma of the names of Senac functions of $1+ma$ real variables returning the values of the derivatives of the fitting function with respect to each of the parameters,
covar is a floating point matrix representing the covariance,
alpha is a floating point matrix representing the curvature,
chisq is a floating point number representing the value of chi-square,
alamda is a real number which initially should be negative and on output will decrease from the value of 0.001d0 by a factor of 10 if a step is successful. If not the variable will grow by a factor of 10. This routine should be called repeatedly until convergence is achieved and then one more time with alamda set to 0d0.

Example:

The sample function here is

$$f(x) = 5exp(-((x-2.0)/3)^2) + 2exp(-((x-5.0)/3)^2).$$

```
-> f(x) := 5*exp(-((x-2.0)/3)^2) + 2*exp(-((x-5.0)/3)^2);
-> fa(x,a1,a2,a3,a4,a5,a6) := a1*exp(-((x-a2)/a3)^2) + a4*exp(-((x-a5)/a6)^2);
-> which corresponds to the coefficient list : [5,2,3,2,5,3] #
-> guess : [4.5, 2.2, 2.8, 2.5, 4.9, 2.8];
-> lista : [1,2,3,4,5,6];
-> mfit : 6;
-> unbind(f1,f2,f3,f4,f5,f6,a1,a2,a3,a4,a6,a6,x);
-> defev(f1, [x,a1,a2,a3,a4,a5,a6], diff(fa(x,a1,a2,a3,a4,a5,a6),a1));
-> defev(f2, [x,a1,a2,a3,a4,a5,a6], diff(fa(x,a1,a2,a3,a4,a5,a6),a2));
-> defev(f3, [x,a1,a2,a3,a4,a5,a6], diff(fa(x,a1,a2,a3,a4,a5,a6),a3));
-> defev(f4, [x,a1,a2,a3,a4,a5,a6], diff(fa(x,a1,a2,a3,a4,a5,a6),a4));
-> defev(f5, [x,a1,a2,a3,a4,a5,a6], diff(fa(x,a1,a2,a3,a4,a5,a6),a5));
-> defev(f6, [x,a1,a2,a3,a4,a5,a6], diff(fa(x,a1,a2,a3,a4,a5,a6),a6));
-> x : 0.1*listn(1,100);
-> y : map(f,x);
-> sig : const(0.02,length(x));
-> alam : -1.0d0;
-> mrqmin(x,y,sig,guess,lista,mfit,fa,[f1,f2,f3,f4,f5,f6],alam)?
 | 83342.04   27053.13   47823.78  59844.33  -41029.08  38569.93 |
 |                                                               |
 | 27053.13   148017.77  73624.79  89954.10  1420.77    5751.73  |
 |                                                               |
 | 47823.78   73624.79   91459.41  86963.43  7275.34    25557.27 |
[|                                                               |,
 | 59844.33   89954.10   86963.43  96098.22  196.94     35493.81 |
 |                                                               |
 | -41029.08  1420.77    7275.34   196.94    53234.94   765.85   |
 |                                                               |
 | 38569.93   5751.73    25557.27  35493.81  765.85     37914.05 |
| 83342.04   27053.13   47823.78  59844.33  -41029.08  38569.93 |
|                                                               |
| 27053.13   148017.77  73624.79  89954.10  1420.77    5751.73  |
|                                                               |
| 47823.78   73624.79   91459.41  86963.43  7275.34    25557.27 |
|                                                               |, 4.62,
| 59844.33   89954.10   86963.43  96098.22  196.94     35493.81 |
|                                                               |
| -41029.08  1420.77    7275.34   196.94    53234.94   765.85   |
|                                                               |
| 38569.93   5751.73    25557.27  35493.81  765.85     37914.05 |
1.89, 2.99, 2.30, 4.74, 3.07], 405.07, 0.00]
```

medfit

Description:

This function fits data to a straight line robustly using the least absolute standard deviation.

$$\mathbf{medfit(x,\ y)} \longrightarrow [\mathbf{a},\ \mathbf{b},\ \mathbf{abdev}]$$

Where:

x is a real list representing the first coordinates of a set of data points,
y is a real list representing the second coordinates of a set of data points,
a is the constant term in the fitted equation $y = a + bx$,
b is the coefficient of x in the fitted equation,
abdev is a floating point number representing the mean absolute deviation in y of the experimental points from the fitted line.

Example:

Here we go back to the sample function used for `fit` at the beginning of this section, viz. $y = 3x - 5$.

```
-> x : [-1,0,1,2,3,4,5,6,7,8,9];
-> y : 3*x - const(5,length(x))?
[-8, -5, -2, 1, 4, 7, 10, 13, 16, 19, 22]
-> medfit(x,y)?
[-5.0000000, 3.0000000, 0.0000000]
```