1 lsr.m

```matlab
function [betacoef,V,J,CovB,So2,modelinfo] = lsr(x,y,modelfun,varargin)
% LSR Least Squares Regression For Linear, Nonlinear, Robust, and Total LSR
%   LSR(X,Y,MODELFUN,VARARGIN) performs a least squares regression to
%   estimate the beta coefficients.  The Jacobian and partial derivatives
%   are computed using finite differencing.  This function mimics the
%   performance on NLINFIT without the reliance on the Statistics and
%   Machine Learning Toolbox.  This function also adds functionality to:
%    o perform linear least squares
%    o detect linearity from modelfun using finite differencing Hessian
%    o perform total least squares
%    o input analytical jacobians
%    o automatically perform chi2 goodness of fit test
%    o disable automatic covariance scaling
%    o add an estimate of the beta coefficients as observation equations
%
%   ∗ See 'doc/doclsr.pdf' for more a more detailed description
%   ∗ See 'exampleLsr.m' for more examples
%
% Inputs:
%   — x              : Predictor variables
%   — y              : Response values
%   — modelfun       : Model function handle @modelfun(betacoef,x)
%   — betacoef0      : Initial regression coefficient values
%
% Optional Parameters:
%   — 'betacoef0'       : Initial regression coefficient values
%   — 'type'            : Type of Regression
%   — 'weights'         : Vector (weights) or Covariance matrix
%   — 'betaCoef0Cov'    : Covariance of beta0 coefficient values
%   — 'JacobianYB'      : Function @(b,x) for Jacobian wrt betacoef
%   — 'JacobianYX'      : Function @(b,x) for Jacobian wrt x
%   — 'scaleCov'        : boolean (default:1) to scale covariance matrix
%   — 'chi2alpha'       : Alpha values for significance
%   — 'RobustWgtFun'    : Robust Weight Function
%   — 'Tune'            : Robust Weight Tuning Function
%   — 'RobustThresh'    : Threshold for Robust Iterations
%   — 'RobustMaxIter'   : Maximum iterations in Robust Least Squares
%   — 'maxiter'         : Maximum iterations for Nonlinear
%   — 'verbose'         : True/False print verbose output to screen
%   — 'DerivStep'       : Difference for numerical Jacobian
%   — 'enforceValidCovB': Attempts to iteratively enforce a valid CovB
%
% Outputs:
%   — betacoef    : Estimated regression coefficients
%   — V           : Residuals
%   — J           : Jacobian with respect to B of the final iteration
%   — CovB        : Estimated Variance Covariance Matrix
%   — So2         : Mean Squared Error (Computed Reference Variance)
%   — modelinfo   : Information about error model in structure
```

1.1 Motivation/Concept

The function *lsr.m* is a standalone matlab script that was written to perform least squares regression. Matlab has built in functions *lscov.m* for linear regression and *nlinfit.m* for nonlinear regression. The MATLAB *curve fitting toolbox* also has some optimization and regression fitting algorithms built in. *lsr.m* is intended to parallel the syntax of nlinfit, with the additional functionality to:

- perform total least squares
- perform linear least squares
- automatically detect linearity of the modelfun using numerical Hessian
- input analytical Jacobians
- perform χ^2 Goodness of fit test to determine the significance of the computed reference variance σ_0^2
- disable covariance matrix scaling
- add an estimate of the beta coefficients as observation equations

Linear and Nonlinear Least Squares are used for systems of equations where the error is only in the dimension of the response variable and the optional stochastic model is only based on the uncertainty in the response variable. These methods are not robust to outliers.

Total Least Squares is used for linear and nonlinear systems of equations where the error is not solely in the dimension of the response variable. When there is no stochastic model, this is also called orthogonal least squares, as the errors are perpendicular to the fit. This method is useful when each of the measurements in the predictor variables have associated uncertainties or covariances. This method is essentially iteratively re-weighting the system of equations by propagating uncertainty in each equation using GLOPOV to determine an estimated uncertainty in the response variable of each equation. This method is not robust to outliers.

Robust Least Squares is used for linear and nonlinear systems of equations where there are likely outliers. A stochastic model may not be input. A weight function iteratively re-weights the solution to reduce the influence of outliers. Robust Least Squares does NOT provide a covariance and associated computed reference variance. Matlab's nlinfit does, but I was uncomfortable duplicating that code because I didn't fully understand the assumptions being made.

1.2 Math

The observation equations are input as a function handle defined by the equation:

$$y = F(\beta, x)$$

Where:

$$\begin{aligned} y &= \text{Response Variables} \\ F &= \text{model function } (\textit{function } y = \textit{modelfun(b,x)} \text{ in matlab}) \\ \beta &= \text{Estimated Regression Coefficients} \\ x &= \text{Predictor Variables} \end{aligned}$$

With Residuals(V) to account for uncertainty:

$$V = F(\hat{\beta}, x) - y$$

The Least Squares Solution minimizes the sum of the square of the residuals:

$$min \sum_{i=1}^{n} v_i^2$$

A stochastic model is introduced as a weight vector or a covariance matrix, where:

$$\begin{cases} \text{No Weights or Covariance} & W = I \\ \text{Weight Vector } (w) & W = diag(w) \\ \text{Covariance Matrix of Response Variables}(\Sigma_{yy}) & W = inv(\Sigma_{yy}) \end{cases}$$

Least Squares Assumptions Made

- The system of equations is over-constrained
- Errors are normally distributed
- There are no outliers (Thought Robust attempts to handle them)
- Errors are not correlated with the magnitude of any of the predictor variables (what is this called?)
- others to add here?

Automatic Partial Derivative Computation

The finite difference method is used to numerically compute the Jacobian Matrix of the model function with respect to the regression coefficients. For Total Least Squares, the Jacobian of the model function with respect to the predictor variables is also computed. The central finite difference is used, with the default h equal to $eps^{\frac{1}{3}}$. For increased accuracy, the user can input function handles for the $JacobianYB$ and $JacobianYX$, which eliminates the need for the numerical derivative calculations.

Automatic Determination of Model Function Linearity

For linear functions, all of the elements of the Hessian should be equal to 0. The Hessian is computed by performing the second partial numeric derivatives with respect to the regression coefficients for each observation, and comparing the values to 0. If all of the second derivatives are equal to 0, the model is linear. As there could potentially be errors with this method, the user can also explicitly input the type of least squares to be performed.

Optional inclusion of β estimate as a set of observation equations

Sometimes, you can directly measure or estimate the predicted beta coefficients and their associated covariances. For example, when attempting to triangulate a point using triginometry you may also have a low quality GPS coordinate and an associated covariance of that point. This function allows you to pass in a covariance matrix for the beta parameter, which automatically adds it as an observation equation to further constrain the solution.

In a simpler example, lets say we had 3 points with (x,y) coordinates and with a standard deviation of 5 in the y dimension. We want to compute the slope and y intercept using $y = mx + b$. The observation equations would look like:

$$\begin{aligned} F_1 : &\quad y_1 + v_1 = mx_1 + b \\ F_2 : &\quad y_2 + v_2 = mx_2 + b \\ F_3 : &\quad y_3 + v_3 = mx_3 + b \end{aligned}$$

But, somehow we also know that the slope should be 1 ± 2 and the y intercept should be 0.0 ± 4. So we add two observation equations with the predicted values as extra response variables:

$$\begin{aligned} F_4 : &\quad m_{est} + v_4 = m \\ F_5 : &\quad b_{est} + v_5 = b \end{aligned}$$

The A Matrix(Jacobian with respect to the regression coefficients) and the Covariance Matrix become:

$$J_{y\beta} = A = \begin{bmatrix} \frac{\partial F_1}{\partial m} & \frac{\partial F_1}{\partial b} \\ \frac{\partial F_2}{\partial m} & \frac{\partial F_2}{\partial b} \\ \frac{\partial F_3}{\partial m} & \frac{\partial F_3}{\partial b} \\ \frac{\partial F_4}{\partial m} & \frac{\partial F_4}{\partial b} \\ \frac{\partial F_5}{\partial m} & \frac{\partial F_5}{\partial b} \end{bmatrix} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \Sigma_{yy} = \begin{bmatrix} \sigma^2_{y_1} & 0 & 0 & 0 & 0 \\ 0 & \sigma^2_{y_2} & 0 & 0 & 0 \\ 0 & 0 & \sigma^2_{y_3} & 0 & 0 \\ 0 & 0 & 0 & \sigma^2_{m_{est}} & 0 \\ 0 & 0 & 0 & 0 & \sigma^2_{b_{est}} \end{bmatrix} = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{bmatrix}$$

χ^2 Goodness of Fit Test (is σ_0^2 statistically equal to 1?)

A two tailed χ^2 Goodness of Fit Test is used to determine if there is an issue with your least squares adjustment at an α significance level. α values are normally low (eg. 0.01, 0.05).

$$\begin{aligned} \text{Null Hypothesis } H_0 &: S_0^2 = 1 \\ \text{Alternative Hypothesis } H_a &: S_0^2 \neq 1 \\ \text{Significance } &: \alpha \end{aligned}$$

Test Statistic:

$$\chi^2 = \frac{vS_0^2}{\sigma^2} = \frac{dof \times S_0^2}{1} = dof \times S_0^2$$

Rejection Region:

$$\begin{aligned} \chi^2 &< \chi^2_{(\alpha/2,dof)} = \chi^2_{low} \\ \chi^2 &> \chi^2_{(1-\alpha/2,dof)} = \chi^2_{high} \end{aligned}$$

Linear Least Squares

The equations to solve the estimated regression coefficient, $\hat{\beta}$, are given using:

$$\begin{aligned} A = J_{y\beta} &= \text{Partial Derivative of F with respect to } \beta \\ WA\hat{\beta} &= Wy + WV \\ \hat{\beta} &= (A^T WA)^{-1} A^T Wy + WV \end{aligned}$$

Additional Linear Least Squares Parameters are computed using the following equations:

$$\begin{aligned} \text{Number of observations} &= m \\ \text{Number of Regression Coefficients} &= n \\ \text{Degrees of freedom (\# of redundant observations)} &= dof = m - n \\ \text{Weighted Residuals} &= WV = WA\hat{\beta} - Wy \\ \text{Reference Variance} &= \sigma_0^2 = \frac{V^T WV}{dof} \\ \text{Cofactor Matrix} &= Q_{yy} = inv(A^T WA) \\ \text{Covariance Matrix of Unkowns} &= \Sigma_{yy} = \sigma_0^2 \times Q_{yy} \\ \text{Covariance Matrix of Observations} &= \Sigma_{\hat{l}\hat{l}} = A\Sigma_{yy}A^T \\ \text{Standard Deviation of Solved Unknowns} &= \sigma_{\hat{\beta}} = \sqrt{diag(\Sigma_{yy})} \\ \text{Predicted y} &= \hat{y} = A\hat{\beta} = F(\hat{\beta}, x) \\ \text{R}^2 \text{ (model skill)} &= \frac{var(\hat{y})}{var(y)} \\ \text{RMSE } &= \sqrt{\frac{VV^T}{m}} \end{aligned}$$

Nonlinear Loop Criteria

For Nonlinear Least Squares the solution is solved iteratively using an initial estimate for the β coefficients. The algorithm is coded to loop until either:

a) the maximum iterations have been exceeded (default = 100)

b) the computed reference variance σ_0^2 increases after an iteration, indicating a small change due to computer numerical precision. *Note that this increase can also occur when the initial β estimate is poor, of there is an error within the input data or model function.

Nonlinear Least Squares

The loop equations to solve the estimated regression coefficient, $\hat{\beta}$, are given using:

$$
\begin{aligned}
i &= \text{while loop iteration number} \\
K &= y - F(x, \widehat{\beta_{i-1}}) \\
WJ_{y\beta}\Delta\hat{\beta} &= WK + WV \\
\Delta\hat{\beta} &= (J_{y\beta}^T W J_{y\beta})^{-1} J_{y\beta}^T W K \\
\hat{\beta}_i &= \hat{\beta}_{i-1} + \Delta\hat{\beta} \\
V &= y - F(x, \hat{\beta}_i) \\
\sigma_0^2 &= \frac{V^T W V}{dof}
\end{aligned}
$$

Additional Nonlinear Least Squares Parameters are computed using the following equations:

$$
\begin{aligned}
\text{Number of observations} &= m \\
\text{Number of Regression Coefficients} &= n \\
\text{Degrees of freedom (\# of redundant observations)} &= dof = m - n \\
\text{Weighted Residuals} &= WV = WJ_{y\beta}\hat{\beta} - Wy \\
\text{Reference Variance} &= \sigma_0^2 = \frac{V^T W V}{dof} \\
\text{Cofactor Matrix} &= Q_{\beta\beta} = inv(J_{y\beta}^T W J_{y\beta}) \\
\text{Covariance Matrix of Unkowns} &= \Sigma_{\beta\beta} = \sigma_0^2 \times Q_{yy} \\
\text{Standard Deviation of Solved Unknowns} &= \sigma_{\hat{\beta}} = \sqrt{diag(\Sigma_{yy})} \\
\text{Predicted y} &= \hat{y} = F(\hat{\beta}, x) \\
\text{R}^2 \text{ (model skill)} &= \text{Not valid for nonlinear least squares} \\
\text{RMSE } &= \sqrt{\frac{VV^T}{m}}
\end{aligned}
$$

Total Least Squares Loop Criteria

The loop criteria for Total Least Squares is the same as for Nonlinear Least Squares.

Total Least Squares

The loop equations to solve the estimated regression coefficients, $\hat{\beta}$, are given using:

$$\begin{aligned}
V_{xx} &= \text{Residual of each observation} \\
J_{yx} &= \text{Partial Derivative of F with respect to x} \\
J_{yx}V_{xx} + J_{y\beta}\Delta\hat{\beta} &= K \\
\Sigma_{xx} &= \text{Covariance of Predictor Variables} \\
W_{eq} &= (J_{yx}\Sigma_{xx}J_{yx}^T)^{-1} \\
V_{eq} &= \text{Equivalent Residuals} \\
W_{eq}J_{y\beta}\Delta\hat{\beta} &= W_{eq}K + W_{eq}V_{eq} \\
\Delta\hat{\beta} &= (J_{y\beta}^T W_{eq} J_{y\beta})^{-1} J_{y\beta}^T W_{eq} K \\
\hat{\beta}_i &= \hat{\beta}_{i-1} + \Delta\hat{\beta} \\
V_{eq} &= y - F(x, \hat{\beta}_i) \\
\sigma_0^2 &= \frac{V_{eq}^T W_{eq} V_{eq}}{dof}
\end{aligned}$$

Additional Total Least Squares Parameters are computed using the following equations:

$$\begin{aligned}
\text{Number of observations} &= m \\
\text{Number of Regression Coefficients} &= n \\
\text{Degrees of freedom (\# of redundant observations)} &= dof = m - n \\
\text{Residuals} &= V = \Sigma_{xx}J_{yx}^T W_{eq} V_{eq} \\
\text{Reference Variance} &= \sigma_0^2 = \frac{V_{eq}^T W_{eq} V_{eq}}{dof} \\
\text{Cofactor Matrix} &= Q_{\beta\beta} = inv(J_{y\beta}^T W J_{y\beta}) \\
\text{Covariance Matrix of Unkowns} &= \Sigma_{\beta\beta} = \sigma_0^2 \times Q_{yy} \\
\text{Standard Deviation of Solved Unknowns} &= \sigma_{\hat{\beta}} = \sqrt{diag(\Sigma_{\beta\beta})} \\
\text{Predicted y} &= \hat{y} = F(\hat{\beta}, x) \\
\text{R}^2 \text{ (model skill)} &= \text{Not valid for nonlinear least squares} \\
\text{RMSE } &= \sqrt{\frac{V_{eq}V_{eq}^T}{m}}
\end{aligned}$$

Robust Loop Criteria

The computation of the reference variance when performing Robust Least Squares is a bit challenging. There is some literature on it, but it is currently not implemented in this function. Therefore, searching for the first time the computed reference variance increases between iterations is no longer feasible. As a replacement, the while loop exits on the following criteria:

a) the maximum iterations have been exceeded (default = 100)

b) the computed reference variance σ_0^2 changes by a value that is less than a threshold (default = 1e-8).

Robust Least Squares

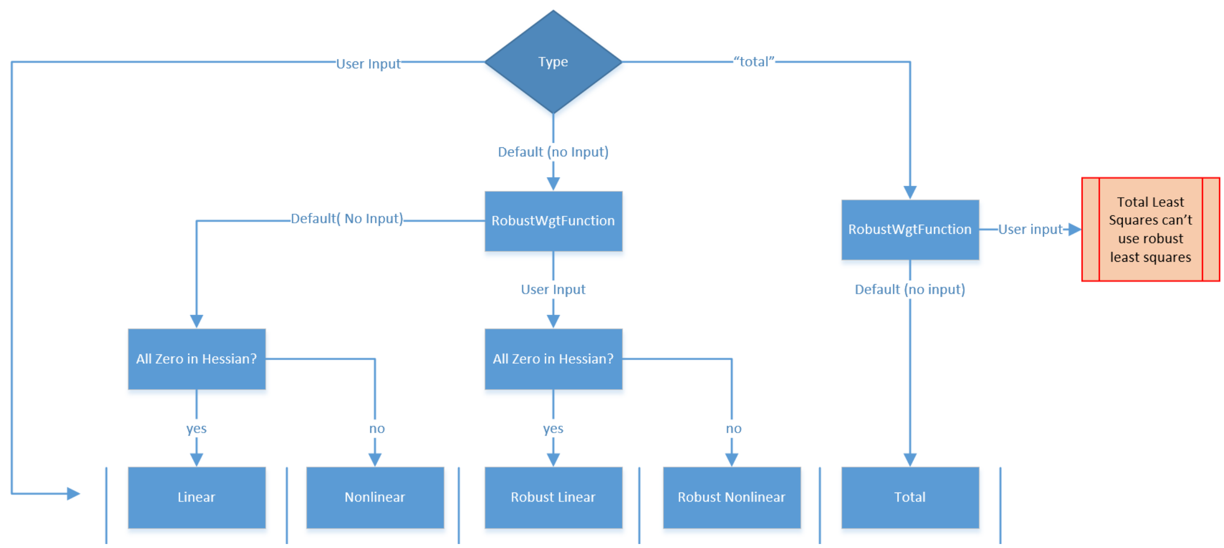
Robust Least Squares uses an iterative re-weighted method where the weight of each observation is determined by how large it's residual is. The first iteration is solved with all weights equal to 1, then the residuals are normalized and converted to a weight using the robust weight function. The normalization of the residuals and weight computations are given by the following equations:

$$
\begin{aligned}
\text{Hat Matrix: } H &= J_{y\beta}(J_{y\beta}^T J_{y\beta})^{-1} J_{y\beta} \\
\text{Leverages: } h &= diag(H) \\
\text{Adjusted Residuals: } V_{adj} &= V * sqrt(h) \\
\text{Ensure minimum residuals: } V_{adj} &= max(1e-6, Vadj) \\
\text{Estimated std (Median Absolute Residuals): } \sigma_{\hat{V_{adj}}} &= MAD(V_{adj})/0.6745 \\
\text{Normalized Residuals: } V_{norm} &= V_{adj}/(RobustTune * \sigma_{V_{adj}}) \\
\text{Weight Vector: } W &= RobustWgtFun(V_{norm}) \\
\beta_i &= \text{Performed with either linear or nonlinear least squares}
\end{aligned}
$$

The statistics from Robust Least Squares solutions should be used with caution. The iterative reweighting methodology effectively removes samples from influencing the dataset. These "outliers" would significantly influence a computed RMSE. Some would argue that Robust Least Squares isn't used as often because it is not "pure" least squares, in this sense, where it "sort of" includes outliers.

1.3 Input Variables

Below is a flowchart and table depicting how the type of least squares is determined and what input parameters apply to each type of least squares.



	Linear	Nonlinear	Robust linear	Robust Nonlinear	Total
x	Required	Required	Required	Required	Required
y	Required	Required	Required	Required	Required
modelfun	Required	Required	Required	Required	Required
betacoef0	no effect w/o betacoef0cov	Required	no effect w/o betacoef0cov	Required	Required
weights	optional	optional	no weights for robust		optional
betacoef0cov	warn if no covariance		no beta covariance for robust		warn if no covariance
JacobianYB	optional	optional	optional	optional	optional
JacobianYX	only applies to total least squares				optional
scaleCov	optional	optional	Does Nothing		optional
chi2alpha	warn if no covariance		Does Nothing		warn if no covariance
Tune	Only Applies to Robust		optional	optional	Only applies to Robust
RobustThresh	Only Applies to Robust		optional	optional	Only applies to Robust
RobustMaxIter	Only Applies to Robust		optional	optional	Only applies to Robust
maxiter	Does Nothing	optional	Does Nothing	optional	optional
verbose	optional	optional	optional	optional	optional
DerivStep	optional	optional	optional	optional	optional

1.3.1 x

Predictor variables

The matrix x is a matrix where each row is an observation, and the number of columns corresponds to the variables in the observation equation.

1.3.2 y

Response values

The column vector y of response values contains a row for each observation equation Size: [Nx1]

1.3.3 modelfun

Model function handle @modelfun(betacoef,x)

The function handle *modelfun* should take a regression coefficient input and x predictor variables matrix as inputs, and output a vector of response values.

1.3.4 betacoef0

Initial β regression coefficient values

1.3.5 type

Type of Regression

A string can be used to explicitly indicate the type of least squares to perform. By default it is assumed that the model is either linear or nonlinear with the distinction computed using a numerical Hessian.

Desired Type	Valid Input Strings
linear	'ols','wls','gls','lin','linear'
nonlinear	'nlin','nonlinear'
robust linear	'robustlinear','robust' (with modelfun Hessian = linear)
robust nonlinear	'robustnonlinear','robust' (with modelfun Hessian = nonlinear)
total	'total','tls'

1.3.6 weights

Vector (weights) or Covariance matrix

A stochastic model is defined by either a vector of weights, or a covariance matrix.

$$\begin{cases} \text{No Weights or Covariance (default)} & W = I \\ \text{Weight Vector } (w) & W = diag(w) \\ \text{Covariance Matrix of Response Variables}(\Sigma_{yy}) & W = inv(\Sigma_{yy}) \end{cases}$$

1.3.7 JacobianYB

Function @(b,x) for Jacobian with respect to the betacoef

A function handle for the Jacobian of the modelfun with respect to the β coefficients. This function is normally not needed, as the numerical partial derivatives do a pretty good job.

1.3.8 JacobianYX

Function @(b,x) for Jacobian with respect to x

A function handle for the Jacobian of the modelfun with respect to the x predictor variables. This function is normally not needed, as the numerical partial derivatives do a pretty good job.

1.3.9 noscale

Boolean (default:0) to scale covariance matrix

By default, most least squares solutions will scale the covariance of the regression coefficients, $\Sigma_{\beta\beta}$, by the computed reference variance, σ_0^2. If you have a high confidence that the initial reference variance is 1, then you can choose to not scale the covariance matrix with this flag. The χ^2 goodness of fit test will give insight into whether or not it is statistically valid to do this.

1.3.10 betaCoef0Cov

Covariance of beta0 coefficient values

The covariance matrix of the initial beta coefficient values is used to add the estimates as observation equations, as described in the Math section above.

1.3.11 chi2alpha

Alpha values for significance

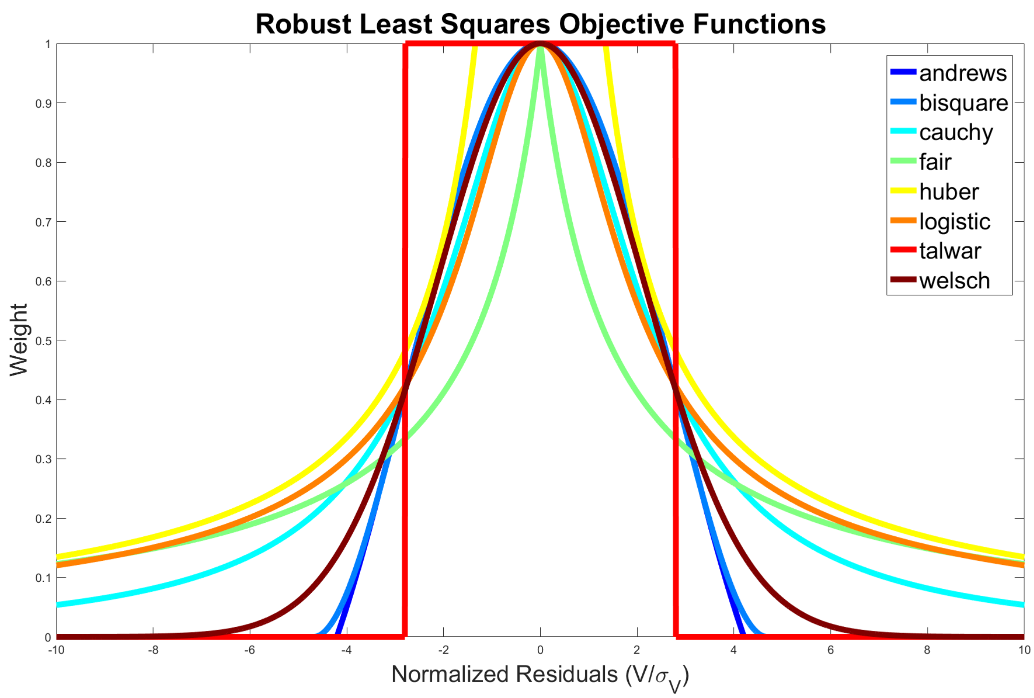
The χ^2 goodness of fit test uses an alpha value for the significance of the test.

1.3.12 RobustWgtFun

Robust Weight Function

The following functions are available as Robust Least Squares functions (Copied Directly From Matlab NLINFIT.M). The user may also pass in a specific function handle instead.

Weight Function	Equation	Default Tuning Constant
'andrews'	$w = I[\lvert r\rvert < \pi] \times \sin(r)/r$	1.339
'bisquare'	$w = I[\lvert r\rvert < 1] \times (1 - r^2)^2$	4.685
'cauchy'	$w = \frac{1}{1+r^2}$	2.385
'fair'	$w = \frac{1}{1+\lvert r\rvert}$	1.400
'huber'	$w = \frac{1}{max(1,\lvert r\rvert)}$	1.345
'logistic'	$w = \frac{\tanh r}{r}$	1.205
'talwar'	$w = I[\lvert r\rvert < 1]$	2.795
'welsch'	$w = e^{-r^2}$	2.985



1.3.13 Tune

Robust Weight Tuning Function

The tune value adjusts the width of the robust function window. A larger tune creates a broader window which will give more weight to higher residuals.

1.3.14 RobustThresh

Threshold for Robust Iterations

The exit criteria for the robust loop is based on the there being a small change in the computed σ_0^2 between iterations that is smaller than the *RobustThresh*. The default value is 1e-8.

1.3.15 RobustMaxIter

Maximum iterations in Robust Least Squares (default = 100)

1.3.16 maxiter

Maximum iterations for Nonlinear (default = 100)

1.3.17 verbose

True/False print verbose output to screen

1.3.18 DerivStep

Difference for numerical Jacobian (default = $eps^{(1/3)}$)

1.3.19 enforceValidCovB

Logical flag to indicate if time should be spent attempting to make CovB a valid covariance matrix

Sometimes the output covariance matrix can be either nonsymmetrical, or non positive semi definite, which indicates an invalid covariance matrix. Basically rounding and other computer math errors results in a covariance matrix that is really really close to valid, but just a bit off. A looping approach that chases its tail somewhat iteratively modifies the matrix to make it symmetrical, then positive semidefinite. These steps sometimes affect the other, and it ends up looping until the math works. It's not the best, but if you really need it, its there.

1.4 Output Variables

1.4.1 betacoef

Estimated regression coefficients

1.4.2 V

Weighted Residuals

1.4.3 J

Jacobian with respect to regression coefficients β

1.4.4 CovB

Estimated Variance Covariance Matrix of regression coefficients β

1.4.5 modelinfo

Extra model data with the following structure:

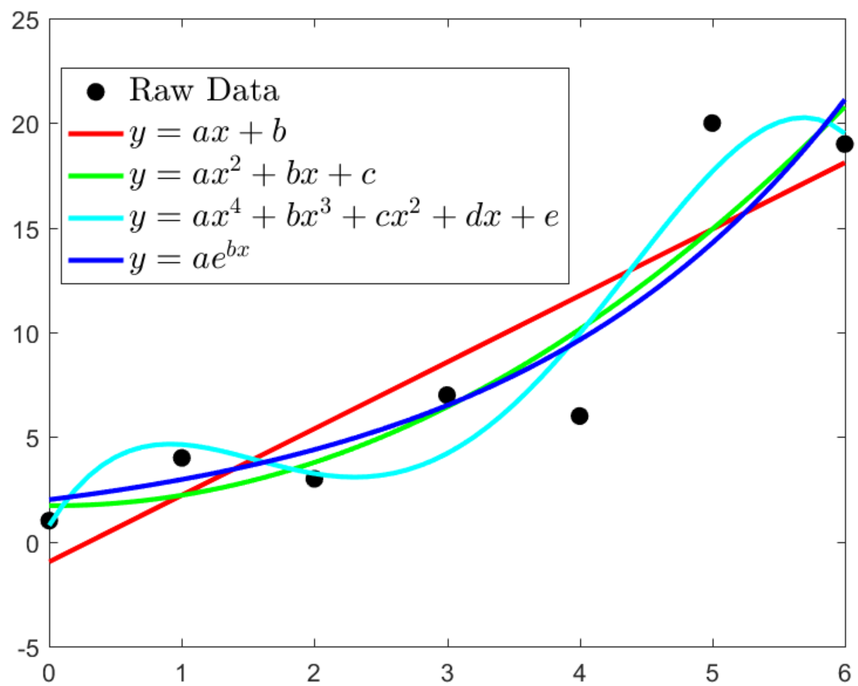
```
modelinfo
├── type (type of regression)
├── nObservationEquations (m)
├── nBetaCoefficients (n)
├── dof (degrees of freedom)
├── betacoef (β̂)
├── V (Weighted Residuals, V_eq for total least squares)
├── R (Same as V for backwards compatability with nlinfit)
├── Vobs (Only for TLS, residual of each observation in x)
├── So2 (σ_0^2)
├── Q (Q_β̂β̂)
├── CovB (Σ_β̂β̂)
├── isCovBscaled (boolean for if CovB was scaled by So2 or not)
├── stdB (σ_β̂ )
├── r2 (r^2)
├── RMSE (Root Mean Square Error)
├── Robust (only exists when robust regression is performed)
│   ├── RobustWgtFun
│   ├── Tune
│   ├── RobustMaxIter
│   ├── RobustThresh
│   └── niter
├── niter (for nonlin or robust, number of iterations in loop)
└── chi2 (option χ^2 tests when covariance matrix input)
    ├── alpha (significance level)
    ├── pass (boolean for if the χ^2 goodness of fit test passed)
    ├── calculatedchi2 (computed χ^2)
    ├── chi2low (χ^2_low)
    ├── chi2high (χ^2_high)
    ├── calculatedSo2 (computed σ_0^2)
    ├── So2low (χ^2_low/dof)
    └── So2high (χ^2_high/dof)
```

Example Usage (*exampleLsr.m*)

Fit Unweighted 2D Data with Different Models

This example demonstrates how to make an anonymous model function and call lsr with various models.

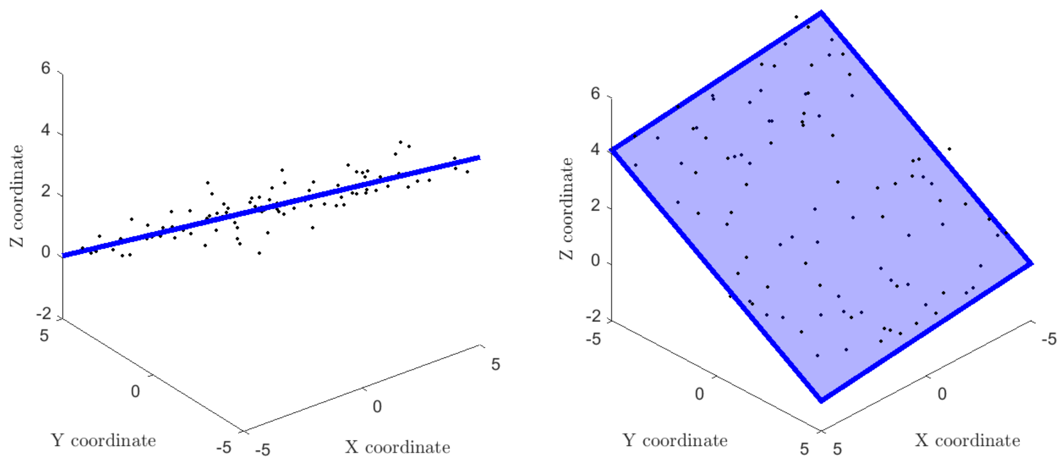
```
%% Fit Different Models to a set of unweighted 2D data
% raw data
x = [0 1 2 3 4 5 6]';
y = [1 4 3 7 6 20 19]';

% Linear Trend  y = mx+b
modelfunLinear = @(b,x) b(1)*x + b(2);
betacoefLinear = lsr(x,y,modelfunLinear);

% 2nd Order Polynomial   y = ax^2+bx+c
modelfunPoly2 = @(b,x) b(1)*x.^2 + b(2)*x +b(3);
betacoefPoly2 = lsr(x,y,modelfunPoly2);

% 4th Order Polynomial  y = ax^4+bx^3+cx^2+dx+e
modelfunPoly4 = @(b,x) b(1)*x.^4 + b(2)*x.^3 + b(3)*x.^2 + b(4)*x.^1 + b(5);
betacoefPoly4 = lsr(x,y,modelfunPoly4);

% Exponential (NonLinear) y = ae^(-bx)
modelfunExp = @(b,x) b(1)*exp(b(2)*x);
betacoef0 = [3 .5]';
betacoefExponential = lsr(x,y,modelfunExp,betacoef0);
```



Fit a Plane to 3D data

This example demonstrates fitting a plane to 3d points. Notice how the predictor variables are input into matrix x so that each row represents an observation.

```
%% Fit Model to 3D Plane (Unweighted)
% beta = [a b c d]  % ax + by + c = z
modelfun3Dplane = @(b,x)(b(1)*x(:,1) + b(2)*x(:,2) + b(3));

%generate 100 data points with X=[-5 5] Y=[-5 5]
rng(1);
truebeta = [-.1 -.5 2]';
xpts = (rand(100,1)-0.5)*10;
ypts = (rand(100,1)-0.5)*10;
zpts = modelfun3Dplane(truebeta,[xpts ypts]) + randn(100,1)*.5;

% do least squares
x = [xpts ypts];
y = zpts;

betacoefPlane = lsr(x,y,modelfun3Dplane);
```

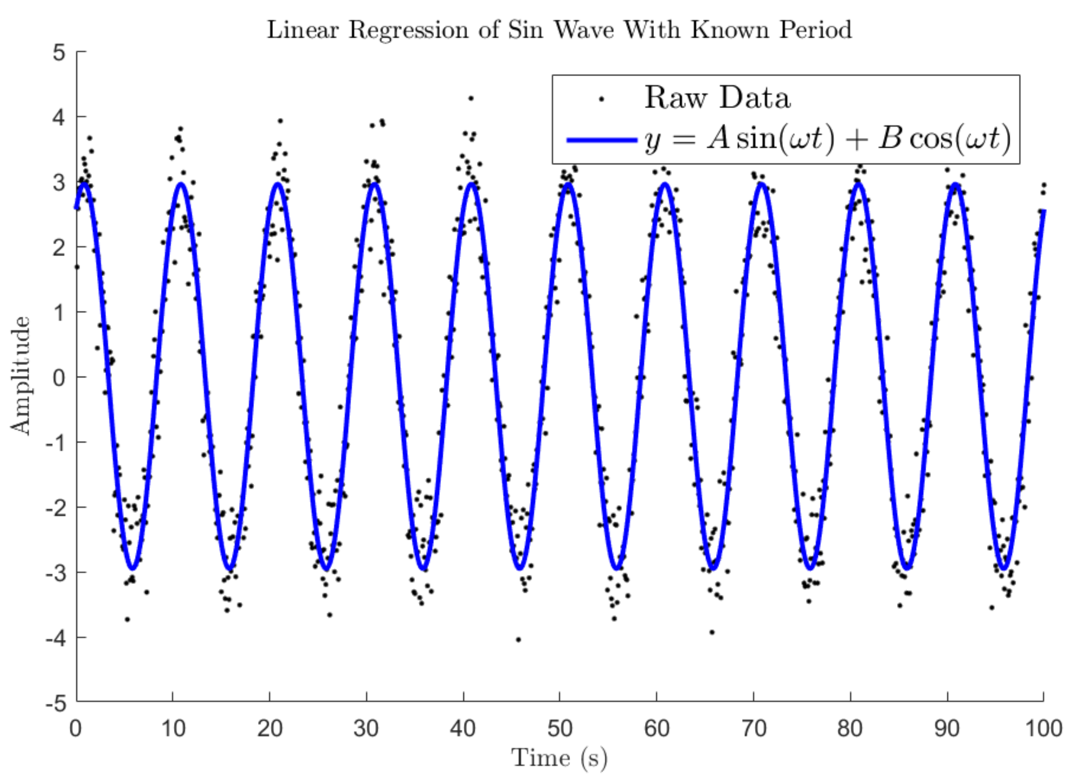


Sin Wave with known period

This example demonstrates how a sine wave can be fit as a linear or nonlinear model. The results will be the same, but there is no need for an initial guess at the true beta in the linear case.

$$\text{NonLinear Function: } y = a\sin(\frac{2\pi}{T}x + \phi)$$
$$\text{Linear Function: } y = a\sin(\frac{2\pi}{T}x) + b\cos(\frac{2\pi}{T}x)$$

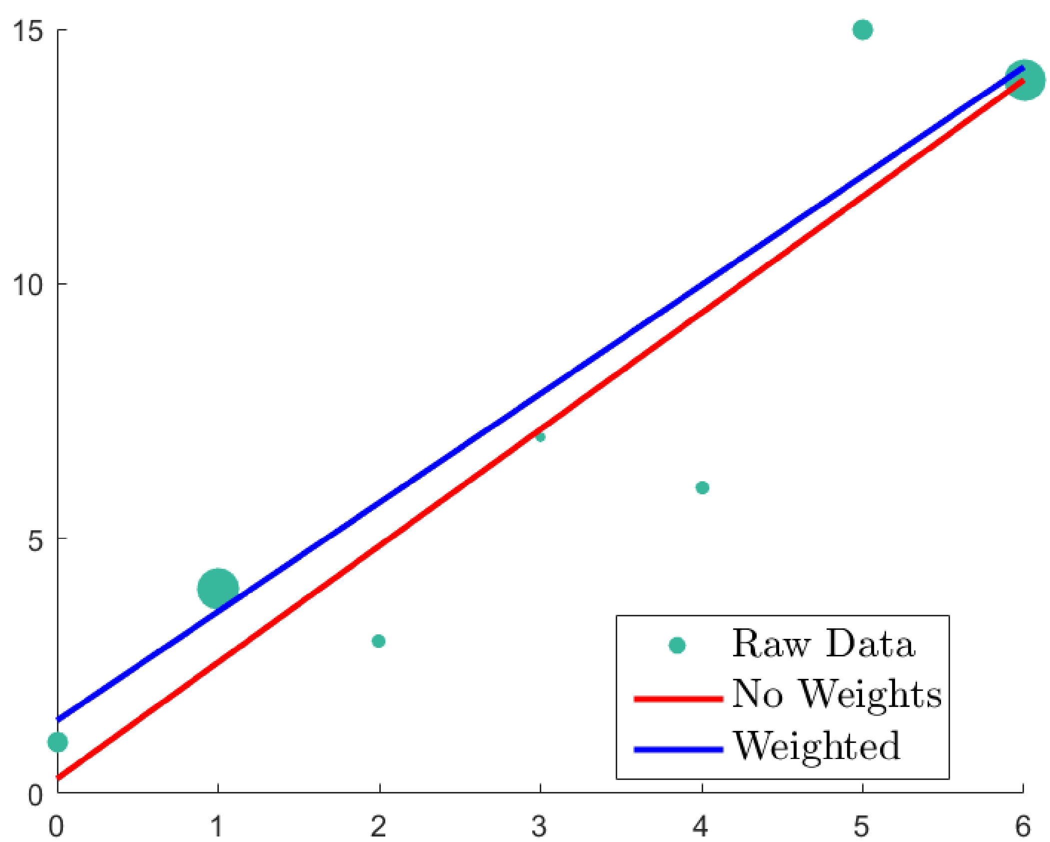
```
%% Sin Wave With Known Period (Nonlinear Observation Equations)
omega = 2*pi/10; %10 second period wave
modelfunSinNonLinear = @(b,x) b(1)*sin(omega*x + b(2));
modelfunSinLinear = @(b,x) b(1)*sin(omega*x)+b(2)*cos(omega*x);

% generate sample data
t = 0:0.1:100;
truebeta = [3 pi/3];
z = modelfunSinNonLinear(truebeta,t)+randn(size(t))*0.5;

% Nonlinear
x = t';
y = z';
betacoefSinNonLinear = lsr(x,y,modelfunSinNonLinear,truebeta,'verbose',true);

% Linear
betacoefSinLinear = lsr(x,y,modelfunSinLinear,'verbose',true);
Amp = sqrt((betacoefSinLinear(1))^2+(betacoefSinLinear(2))^2);
Phi = atan2(betacoefSinLinear(2),betacoefSinLinear(1));
```



Different Ways to Weight Equations

This example demonstrate how to add a stochastic model to the least squares solution using either a weight vector or a covariance matrix. The difference between the weighted solution and unweighted solution for a linear fit is shown in the figure.

```
%% Different ways to weight equations
x = [0 1 2 3 4 5 6]';
y = [1 4 3 7 6 15 14]';
sigmaY = [2 1 3 4 3 2 1];

modelfunLinear = @(b,x) b(1)*x + b(2);
% No weights
betacoefNoWeight = lsr(x,y,modelfunLinear);
% Weights as vector
weightVector = 1./(sigmaY.^2);
betacoefWeightVector = lsr(x,y,modelfunLinear,'Weights',weightVector);
% Weights as Covariance Matrix
covarianceMatrix = diag(sigmaY.^2);
betacoefCovMatrix = lsr(x,y,modelfunLinear,'Weights',covarianceMatrix);
```



2D Conformal Coordinate Transformation

This example demonstrates how you can have a model function which handles multiple observation equations from the same set of data. Notice that the function is also externally defined.

The 2D Conformal equations are:

$$X = (S\cos(\theta))x - (S\sin(\theta))y + T_x$$
$$Y = (S\sin(\theta))x + (S\cos(\theta))y + T_y$$

By substituting:

$$a = S\cos(\theta)$$
$$b = S\sin(\theta)$$

Where:

$$\theta = \tan^{-1}(\frac{b}{a})$$
$$S = \frac{a}{\cos(\theta)}$$

The observation equations become:

$$\begin{aligned} F: &\qquad X = ax - by + T_x \\ G: &\qquad Y = bx + ay + T_y \end{aligned}$$

Note that every $(X, Y) \rightarrow (x, y)$ correspondence produces 2 observation equations.

```
%% 2D Conformal Transformation
function y = conformal2d(beta,x)
% 2D conformal coordinate transformation (beta = [a;b;c;d], x = [xc(:) yc(:)])
nObservations = size(x,1);
y = nan(nObservations*2,1);
y(1:2:end) = beta(1)*x(:,1) - beta(2)*x(:,2) + beta(3);
y(2:2:end) = beta(2)*x(:,1) + beta(1)*x(:,2) + beta(4);
end
```

```
%% 2D Conformal Transformation with covariances (Linear 2 Equations per Observation)
x_coord2 = [1 2 3]; y_coord2 = [0 5 1];  % raw data 'to'
x_coord1 = [6 1 8]; y_coord1 = [3 12 8]; % raw data 'from'

Sc = [0.5 0.3 0 0 0 0;
    0.3 0.5 0 0 0 0;
    0 0 0.4 0.1 0 0;
    0 0 0.1 0.2 0 0;
    0 0 0 0 0.7 -0.4;
    0 0 0 0 -0.4 0.4]; %variance-covariance of data2

modelfun = @conformal2d;
y = nan(2*numel(x_coord1),1);
y(1:2:end)=x_coord2;
y(2:2:end)=y_coord2;

x = [x_coord1' y_coord1'];

betacoef2DConformal = lsr(x,y,modelfun,'Weights',Sc,'verbose',true);
```

Unweighted 3D Conformal Transformation

This example demonstrates a more complex 7-parameter 3D conformal transformation governed by the following equations:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} \cos(\kappa)\cos(\phi) & \cos(\phi)\sin(\kappa) & -\sin(\phi) \\ \cos(\kappa)\sin(\omega)\sin(\phi) - \cos(\omega)\sin(\kappa) & \cos(\kappa)\cos(\omega) + \sin(\kappa)\sin(\omega)\sin(\phi) & \cos(\phi)\sin(\omega) \\ \sin(\kappa)\sin(\omega) + \cos(\kappa)\cos(\omega)\sin(\phi) & \cos(\omega)\sin(\kappa)\sin(\phi) - \cos(\kappa)\sin(\omega) & \cos(\omega)\cos(\phi) \end{bmatrix}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = S \times R \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$

Notice that a helper function is used to combine the X,Y,Z output from conformal 3D into a vector. This satisfies the requirement that the response variables are in a column vector for input to *lsr*.

```
%% 3D Conformal Transformation
% 3D Conformal Transformation
function [X,Y,Z] = conformal3d(S,omega,phi,kappa,Tx,Ty,Tz,x,y,z)
Rx = [1 0 0; 0 cos(omega) sin(omega); 0 -sin(omega) cos(omega)];
Ry = [cos(phi) 0 -sin(phi); 0 1 0;sin(phi) 0 cos(phi)];
Rz = [cos(kappa) sin(kappa) 0; -sin(kappa) cos(kappa) 0; 0 0 1];
R = Rx*Ry*Rz;

XYZ = S * R * [x(:)';y(:)';z(:)'] + repmat([Tx; Ty; Tz],1,numel(x));

X = XYZ(1,:)';
Y = XYZ(2,:)';
Z = XYZ(3,:)';
end

% 3D Conformal Transformation Combined to Vector output
function y = conformal3dfun(S,omega,phi,kappa,Tx,Ty,Tz,x)
[X,Y,Z] = conformal3d(S,omega,phi,kappa,Tx,Ty,Tz,x(:,1),x(:,2),x(:,3));
y = [X Y Z]';
y = y(:);
end
```

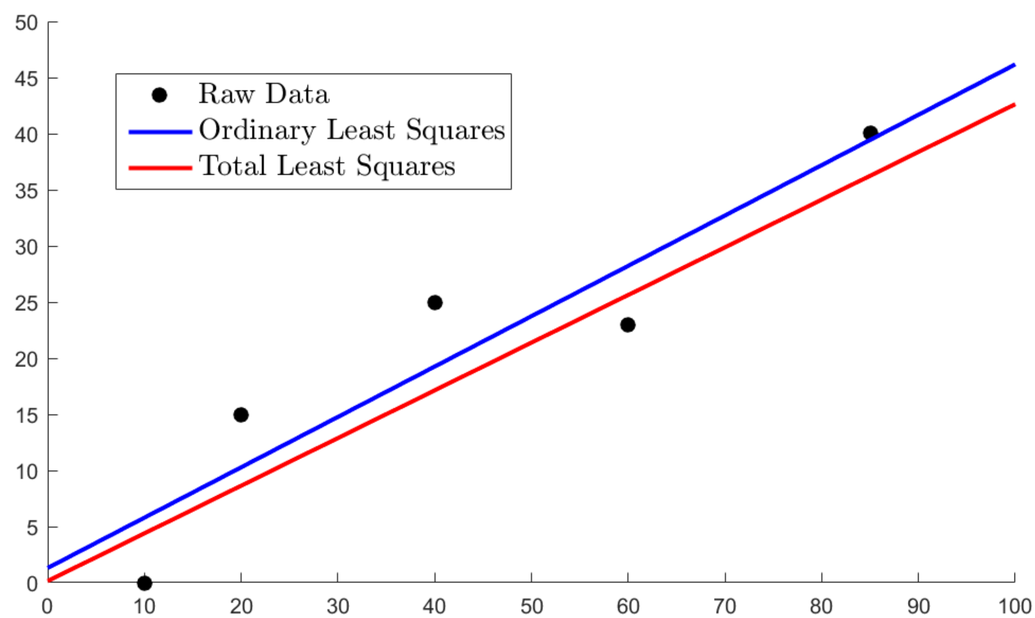
Notice that using anonymous functions, it is very easy to choose which beta parameters to solve for. In this example, the scale was set to 1, and the other 6 parameters were set as the regression coefficients.

```
%% Unweighted 3D Conformal Transformation (Nonlinear 3 Equations per observation)
modelConformal = @(b,x) conformal3dfun(b(1),b(2),b(3),b(4),b(5),b(6),b(7),x);
% Here the scale is fixed == 1, and not solved as a beta coefficient
modelConformalFixScale = @(b,x) conformal3dfun(1,b(1),b(2),b(3),b(4),b(5),b(6),x);

%generate data
xpts = (rand(10,1)-0.5)*100;
ypts = (rand(10,1)-0.5)*100;
zpts = (rand(10,1)-0.5)*100;
x = [xpts ypts zpts];

truebeta = [1 pi/2 pi pi/4 2 3 4]';
XYZ = modelConformal(truebeta,x) + randn(3*numel(xpts),1);
Xpts = XYZ(1:3:end);
Ypts = XYZ(2:3:end);
Zpts = XYZ(3:3:end);

% do least squares
y = [Xpts Ypts Zpts]';
y = y(:);
betacoef0 = truebeta;
betacoef3Dconformal = lsr(x,y,modelConformal,betacoef0,'verbose',true);
betacoef3Dconformal2 = lsr(x,y,modelConformalFixScale,betacoef0(2:end),'verbose',true);
```

Linear Line with Total Least Squares

This example shows how to use total least squares. The plot depicts how weighting a regression with total least squares is different than an unweighted solution, which should be fairly intuitive. This is a simple example, however, and total least squares has advantages that are beyond the scope of this example.

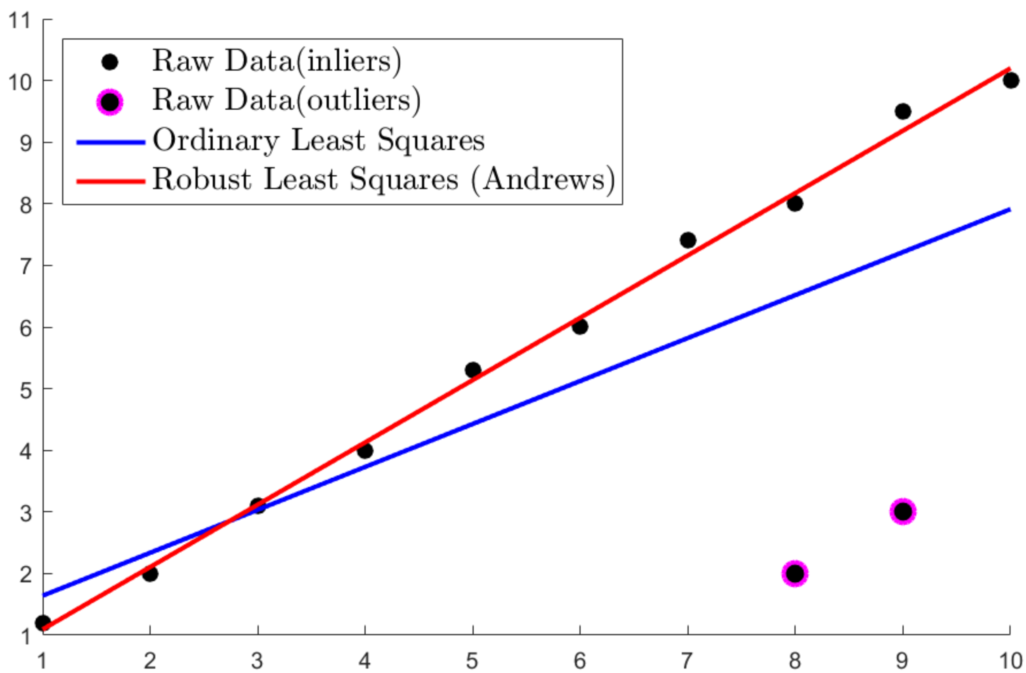
```
%% Linear Line with Total Least Squares
xpts = [10 20 60 40 85];
ypts = [0 15 23 25 40];
covxy = blkdiag([45 -30;-30 30],[20 -10;-10 70],[80 4;4 4],[40 -13;-13 60],[30 -25;-25 30]);

modelfunLinear = @(b,x) b(1)*x + b(2);
modelfunLinearTLS = @(b,x) b(1)*x(:,1) + b(2) -x(:,2);

% ordinary least squares
x = xpts';
y = ypts';
[betacoefOrdinary,V,J,CovB,So2,modelinfo]= lsr(x,y,modelfunLinear);

% total least squares
x = [xpts' ypts'];
y = zeros(5,1);
beteacoef0 = betacoefOrdinary;
[betacoefTLS,V2,J2,CovB2,So22,modelinfo2] = lsr(x,y,modelfunLinearTLS,beteacoef0,'type','tls
    ','Weights',covxy);
```



Robust Least Squares for Line with outliers

This example demonstrates how robust least squares is robust to outliers.

```
%% Robust Least Squares for Line with outliers
% data has one outlier (8,12)
x = [1 2 3 4 5 6 7 8 9 10 8 9]';
y = [1.2 2 3.1 4 5.3 6 7.4 8 9.5 10 2 3]';
modelfunLinear = @(b,x) b(1)*x + b(2);

% Ordinary Least Squares
betacoefOrdinary = lsr(x,y,modelfunLinear,'verbose',true);

% Robust Least Squares
betacoefRobust = lsr(x,y,modelfunLinear,'RobustWgtFun','andrews','verbose',true);
```



χ^2 Goodness of Fit Test for linear line, Don't Scale Covariance

This example demonstrates how the χ^2 Goodness of Fit Test can be used to see if it is valid to not scale the covariance matrix. In this example, it is valid to not scale the covariance matrix. The effect of poor estimation of input uncertainty and the effect of not scaling the covariance matrix are shown in the table shown.

```matlab
%% Chi2 Test for linear line, Dont Scale Covariance
% generate data
rng(2)
stdy = 5;
x = (0:1:50)';
modelfunLinear = @(b,x) b(1)*x + b(2);
truebeta = [1 2];
y = modelfunLinear(truebeta,x)+randn(size(x))*stdy;
CovarianceMatrix = diag(ones(size(x)))*stdy.^2;

% Linear With Covariance Scaled Correctly
[betacoefA,~,~,CovB_A,MSEA,ErrorModelInfoA] = lsr(x,y,modelfunLinear,'Weights',
    CovarianceMatrix);
% Linear With Covariance Scaled High (Overestimating Errors)
[betacoefB,~,~,CovB_B,MSEB,ErrorModelInfoB] = lsr(x,y,modelfunLinear,'Weights',
    CovarianceMatrix*5);
% Linear With Covariance Scaled Low (UnderEstimating Errors)
[betacoefC,~,~,CovB_C,MSEC,ErrorModelInfoC] = lsr(x,y,modelfunLinear,'Weights',
    CovarianceMatrix/5);
% Linear With Covariance Scaled Correctly and CovB Not Scaled
[betacoefD,~,~,CovB_D,MSED,ErrorModelInfoD] = lsr(x,y,modelfunLinear,'Weights',
    CovarianceMatrix,'scalecov',false);
% Test with fixed chi2alpha
[betacoefE,~,~,CovB_E,MSEE,ErrorModelInfoE] = lsr(x,y,modelfunLinear,'Weights',
    CovarianceMatrix,'chi2alpha',0.10);
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    Covariance Chi2 Test
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Type	BetaCoef(1)	BetaCoef(2)	So2	So2Low	So2High	Chi2 Test
Correct Scale	1.125 ± 0.044	-1.626 ± 1.278	0.857151	0.643978	1.433110	PASS @ 5%
Overestimate Errors	1.125 ± 0.044	-1.626 ± 1.278	0.171430	0.643978	1.433110	FAIL @ 5%
Underestimate Errors	1.125 ± 0.044	-1.626 ± 1.278	4.285756	0.643978	1.433110	FAIL @ 5%
Correct Scale (noscale = true)	1.125 ± 0.048	-1.626 ± 1.380	0.857151	0.643978	1.433110	PASS @ 5%
Correct Scale (chi2alpha = 0.1)	1.125 ± 0.044	-1.626 ± 1.278	0.857151	0.692455	1.353850	PASS @ 10%

Analytical Jacobians

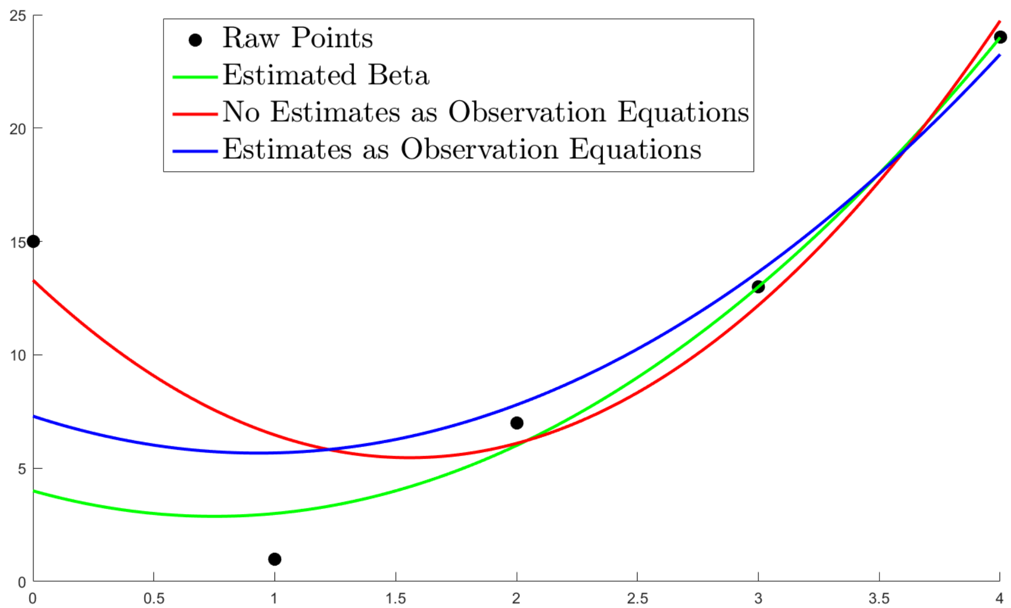
This example demonstrates an example case when analytical Jacobian functions are input. Notice that because this is performing TLS, the t and z values must both be on the same side of the equation, and the resultant y response variables are all equal to 0. The solution from numerical Jacobians and analytical Jacobians are almost exactly the same in this instance (within e-10), though the processing time for the analytical Jacobians was about twice as fast. This speed increase will likely vary depending on the model function.

```
%% Analytical Jacobians
% sine wave with known period
omega = 2*pi/10; %10 second period wave
modelfunSinNonLinear = @(b,x) b(1)*sin(omega*x + b(2));
% need both x and y on same side of equation for TLS
modelfunSinNonLinearTLS = @(b,x) b(1)*sin(omega*x(:,1) + b(2))-x(:,2);
% Partial Derivative With Respect to Beta Regression Coefficients
modelfunSinNonLinearJB = @(b,x) [sin(omega*x(:,1) + b(2)) b(1)*cos(omega*x(:,1) + b(2))];
% Partial Derivative With Respect to Predictor Variables
modelfunSinNonLinearJX = @(b,x) ...
    bumphdiag([b(1)*omega*cos(b(2) + omega*x(:,1)) -ones(size(x(:,2)))],1);

% generate sample data
t = rand(1,100)*100;
truebeta = [3 pi/3];
z = modelfunSinNonLinear(truebeta,t)+randn(size(t))*0.5;

% Nonlinear
x = [t' z'];
y = zeros(size(t'));
betacoefSinNonLinear = lsr(x,y,modelfunSinNonLinearTLS,truebeta,'verbose',true,...
    'type','tls');

betacoefExplicitPartials = lsr(x,y,modelfunSinNonLinearTLS,truebeta,'verbose',true,...
    'JacobianYB',modelfunSinNonLinearJB,'JacobianYX',modelfunSinNonLinearJX,...
    'type','tls');
```

1.4.6 Use Regression Coefficient Estimate as Observation Equations

This example demonstrates how you can use the estimate of the regression coefficients as observation equations to influence the regression model. Notice how the blue line, with the estimated regression coefficients used as observation equations, is closer to the green line, which is the line generated using the estimated regression coefficients.

```
%% Use Regression Coefficient Estimate as Observation Equations
% y = ax^2+bx+c
x = [0 1 2 3 4]';
y = [15 1 7 13 24]';
Syy = diag([1 1.5 1.3 0.3 0.5]);
betaest = [2 -3 4];
Sbb = diag([0.1 0.1 1]);
modelfun = @(b,x) b(1)*x.^2 + b(2)*x + b(3);

betacoef = lsr(x,y,modelfun,betaest,'weights',Syy,'verbose',true);
betacoefEst = lsr(x,y,modelfun,betaest,'weights',Syy,'betaCoef0Cov',Sbb,'verbose',true);
```



Modify the DerivStep for Partial Derivatives

This example demonstrates how you can modify the step used for computing numerical partial derivatives. While this option is available, it should only be used in very rare, specific cases.

```
%% DerivStep
modelConformal = @(b,x) conformal3dfun(b(1),b(2),b(3),b(4),b(5),b(6),b(7),x);

%generate data
xpts = (rand(10,1)-0.5)*100;
ypts = (rand(10,1)-0.5)*100;
zpts = (rand(10,1)-0.5)*100;
x = [xpts ypts zpts];

truebeta = [1 pi/2 pi pi/4 2 3 4]';
XYZ = modelConformal(truebeta,x) + randn(3*numel(xpts),1);
Xpts = XYZ(1:3:end);
Ypts = XYZ(2:3:end);
Zpts = XYZ(3:3:end);

% do least squares
y = [Xpts Ypts Zpts]';
y = y(:);
betacoef0 = truebeta;
betacoef3Dconformal = lsr(x,y,modelConformal,betacoef0,'verbose',true);
betacoef3Dconformal = lsr(x,y,modelConformal,betacoef0,'verbose',true,'derivstep',3);
% No real good reason to change the deriv step, but hey, its there
```

Example of poor observation equations

This example demonstrates a poor observation equation for linear least squares. Notice how using the linear least squares equations to solve the least squares solution, y is nowhere to be found in the solution. In the case of linear least squares, y must be input as a response variable rather than a reactant if it is a constant and not multiplied by any beta coefficients. When it is not multiplied by a beta coefficient, the Jacobian will not contain y, and y will not be factored into the equation.

With Observation Equation:

$$0 = v = mx + b - y$$
$$0 = y = \beta_1 x_1 + \beta_2 - x_2$$

The Jacobian is:

$$J_{yb} = \begin{bmatrix} \dfrac{\partial y}{\partial \beta_1} & \dfrac{\partial y}{\partial \beta_2} \end{bmatrix} = \begin{bmatrix} x_1 & 1 \end{bmatrix}$$

The Hessian is all 0s, which implies linear least squares.

$$H = \begin{bmatrix} \dfrac{\partial^2 y}{\partial \beta_1^2} & \dfrac{\partial^2 y}{\partial \beta_1 \partial \beta_2} \\ \dfrac{\partial^2 y}{\partial \beta_2 \partial \beta_1} & \dfrac{\partial^2 y}{\partial \beta_2^2} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Recall for linear least squares:

$$A = J_{y\beta} = \text{Partial Derivative of F with respect to } \beta$$
$$WA\hat{\beta} = Wy + WV$$
$$\hat{\beta} = (A^T WA)^{-1} A^T Wy + WV$$

Notice that x_2, does not factor into the equation at all! Refer to the nonlinear least squares equations and notice that the K matrix will contain x_2 and therefore it will influence the solution.

```
%% Bad Observation Equationa
% 0 = mx + b - y; %this is a BAD observation equation
%  the observation equation is linear AND y is a reactant and not
%  multiplied by a beta coefficient.  With linear least squares, y now
%  doesnt influence the result and we end up with the null case. b = [0 0];

modelfunbad = @(b,x) b(1)*x(:,1)+b(2)+x(:,2);

% raw data
x = [0 1 2 3 4 5 6]';
y = [1 4 3 7 6 20 19]';

X = [x y];
Y = zeros(size(y));

betacoef = lsr(X,Y,modelfunbad)
% With nonlinear least squares, you can have a reactant variable not
% multipied by a beta coefficient because it ends up in the system of
% equations in the K variable.  BUT you need a guess at beta0coef
betacoefNonLinear = lsr(X,Y,modelfunbad,[-3; 1],'type','nonlinear')
```