LWP

Locally Weighted Polynomials toolbox for Matlab/Octave

ver. 2.2

Gints Jekabsons

E-mail: gints.jekabsons@rtu.lv
http://www.cs.rtu.lv/jekabsons/

User's manual

September, 2016

CONTENTS

1. INTRODUCTION

What is LWP

LWP is a Matlab/Octave toolbox implementing Locally Weighted Polynomial regression (also known as Local Regression / Locally Weighted Scatterplot Smoothing / LOESS / LOWESS and Kernel Smoothing). With this toolbox you can fit local polynomials of any degree using one of the nine kernels with metric window widths or nearest neighbor window widths to data of any dimensionality. A function for optimization of the kernel bandwidth is also available. The optimization can be performed using Leave-One-Out Cross-Validation, GCV, AICC, AIC, FPE, T, S, or separate validation data. Robust fitting is available as well.

Some of the original papers on locally weighted regression methods include (Cleveland et al., 1992; Cleveland & Devlin, 1988; Cleveland, 1979; Stone, 1977; Nadaraya, 1964; Watson, 1964).

This user's manual provides overview of the functions available in the LWP toolbox.

LWP toolbox can be downloaded at http://www.cs.rtu.lv/jekabsons/.

The toolbox code is licensed under the GNU GPL ver. 3 or any later version.

Feedback

For any feedback on the toolbox including bug reports feel free to contact me via the email address given on the title page of this user's manual.

Citing the LWP toolbox

Jekabsons G., Locally Weighted Polynomials toolbox for Matlab/Octave, 2016, available at http://www.cs.rtu.lv/jekabsons/

2. AVAILABLE FUNCTIONS

LWP toolbox provides the following list of functions:

- `lwppredict` – predicts response values for the given query points using Locally Weighted Polynomial regression;
- `lwpparams` – creates configuration for LWP (the output structure is for further use with all the other functions of the toolbox);
- `lwpeval` – evaluates predictive performance of LWP using one of the criteria;
- `lwpfindh` – finds the "best" bandwidth for a kernel.

2.1. Function `lwppredict`

Purpose:

Predicts response values for the given query points using Locally Weighted Polynomial regression. Can also provide the smoothing matrix **L**.

Call:

```
[Yq, L] = lwppredict(Xtr, Ytr, params, Xq, weights, failSilently)
```

All the input arguments, except the first three, are optional. Empty values are also accepted (the corresponding defaults will be used).

Input:

`Xtr, Ytr` : Training data. `Xtr` is a matrix with rows corresponding to observations and columns corresponding to input variables. `Ytr` is a column vector of response values. To automatically standardize `Xtr` to unit standard deviation before performing any further calculations, set `params.standardize` to `true`.
If `Xq` is not given or is empty, `Xtr` also serves as query points.
If the dataset contains observations with coincident `Xtr` values, it is recommended to merge the observations before using the LWP toolbox. One can simply reduce the dataset by averaging the `Ytr` at the tied values of `Xtr` and supplement these new observations at the unique values of `Xtr` with an additional weight.

`params` : A structure of parameters for LWP. See function `lwpparams` for details.

`Xq` : A matrix of query data points. `Xq` should have the same number of columns as `Xtr`. If `Xq` is not given or is empty, query points are `Xtr`.

`weights` : Observation weights for training data (which multiply the kernel weights). The length of the vector must be the same as the number of observations in `Xtr` and `Ytr`. The weights must be nonnegative.

`failSilently` : In case of any errors, whether to fail with an error message or just output `NaN`. This is useful for functions that perform parameter optimization and could try to wander out of ranges (e.g., `lwpfindh`) as well as for drawing plots even if some of the response values evaluate to `NaN`. Default value = `false`. See also argument `safe` of function `lwpparams`.

Output:

`Yq` : A column vector of predicted response values at the query points (or `NaN` where calculations failed).

`L` : Smoothing matrix. Available only if `Xq` is empty.

Remarks:

Locally Weighted Polynomial regression is designed to address situations in which models of global behaviour do not perform well or cannot be effectively applied without undue effort. LWP is a nonparametric regression method that is carried out by pointwise fitting of low-degree polynomials to localized subsets of the data. The advantage of this method is that the analyst is not required to specify a global function. However, the method requires fairly large, densely sampled datasets in order to produce good models and it is relatively computationally intensive.

The assumption of the LWP regression is that near the query point the value of the response variable changes smoothly and can be approximated using a low-degree polynomial. The coefficients of the polynomial are calculated using weighted least squares method giving the largest weights to the nearest data observations and the smallest or zero weights to the farthest data observations.

For further details on kernels, see remarks for function `lwpparams`.

2.2. Function `lwpparams`

Purpose:

Creates configuration for LWP. The output structure is for further use with all the other functions of the toolbox.

Call:

```
params = lwpparams(kernel, degree, useKNN, h, robust, knnSumWeights, standardize, safe)
```

All the input arguments of this function are optional. Empty values are also accepted (the corresponding defaults will be used).

Input:

`kernel` : Kernel type (string). See user's manual for details. Default value = `'TRC'`.
`'UNI'`: Uniform (rectangular)
`'TRI'`: Triangular
`'EPA'`: Epanechnikov (quadratic)
`'BIW'`: Biweight (quartic)
`'TRW'`: Triweight
`'TRC'`: Tricube
`'COS'`: Cosine
`'GAU'`: Gaussian
`'GAR'`: Gaussian modified (Rikards et al., 2006).

`degree` : Polynomial degree. Default value = 2. If `degree` is not an integer, let `degree` = $d_1 + d_2$ where d_1 is an integer and $0 < d_2 < 1$; then a "mixed degree fit" is performed where the fit is a weighted average of the local polynomial fits of degrees d_1 and $d_1 + 1$ with weight $1 - d_2$ for the former and weight d_2 for the latter (Cleveland & Loader, 1996).

`useKNN` : Whether the bandwidth for kernel is defined in nearest neighbors (`true`) or as a metric window (`false`). Default value = `true`.

`h` : Window size of the kernel defining its bandwidth. If `useKNN` = `true`, `h` defines the number of nearest neighbors: for `h` <= 1, the number of nearest neighbors is the fraction `h` of the whole dataset; for `h` > 1, the number of nearest neighbors is integer part of `h`. If `useKNN` = `false`, `h` defines metric

window size. More specifically, let *maxDist* be the distance between two opposite corners of the hypercube defined by `Xtr`, then: for the first 7 kernel types, `h`**maxDist* is kernel radius; for kernel type `'GAU'`, `h`**maxDist* is standard deviation; for kernel type `'GAR'`, `h` is coefficient alpha (smaller alpha means larger bandwidth; see remarks for details). In any case, `h` can also be called smoothing parameter. Default value = 0.5, i.e., 50% of all data if `useKNN` = `true` or 50% of *maxDist* if `useKNN` = `false` and kernel type is not `'GAR'`.
Note that if `useKNN` = `true`, the farthest of the nearest neighbors will get 0 weight.

`robust` : Whether to use robust local regression and how many iterations to use for the robust weighting calculations. Typical values range from 2 to 5. Default value = 0 (robust version is turned off). The algorithm for the robust version is from Cleveland (1979).

`knnSumWeights` : This argument is used only if `useKNN` = `true` and fitting uses observation weights or robustness weights (when `robust` > 0). Set to `true` to define neighborhoods using a total weight content `params.h` (relative to sum of all weights) (Hastie et al., 2009). Set to `false` to define neighborhoods just by counting observations irrespective of their weights, except if a weight is 0. Default value = `true`.

`standardize` : Whether to standardize all input variables to unit standard deviation. Default value = `true`.

`safe` : Whether to allow prediction only if the number of the available data observations for the polynomial at a query point is larger than or equal to the number of its coefficients. Default value = `true`, i.e., the function will fail with error message or output `NaN` (what is the exact action to be taken, is determined by argument `failSilently` of the called function). If `safe` is set to `false`, coefficients will be calculated even if there is rank deficiency. To avoid these situations altogether, make the bandwidth of the kernel sufficiently wide. Note that setting `safe` = `false` is not available in Octave.

Output:

`params` : A structure of parameters for further use with all the other functions of the toolbox containing the provided values (or defaults, if not provided).

Remarks:

Let $b(X)$ be a vector of polynomial terms in X. At each query point $x_0 \in \mathbf{R}^d$ solve

$$\min_{\beta(x_0)} \sum_{i=1}^{n} K_\lambda(x_0, x_i)(y_i - b(x_i)^T \beta(x_0))^2$$

to produce the fit $\hat{f}(x_0) = b(x_0)^T \hat{\beta}(x_0)$ where K is a weighting function or kernel

$$K_\lambda(x_0, x) = D\left(\frac{\|x - x_0\|}{h_\lambda(x_0)}\right)$$

where ||.|| is the Euclidean norm and $h_\lambda(x_0)$ is a width function (indexed by λ) that determines the width of the neighborhood at x_0. For metric window widths, $h_\lambda(x_0) = \lambda$ is constant. For k-nearest neighborhoods, the neighborhood size k replaces λ, and we have $h_k(x_0) = \|x_0 - x_{[k]}\|$ where $x_{[k]}$ is the farthest of the k nearest neighbors x_i to x_0.

The LWP toolbox implements the following kernel functions:

1. Uniform (rectangular)

$$D(u) = \begin{cases} 1 & \text{if } |u| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

2. Triangular

$$D(u) = \begin{cases} 1 - |u| & \text{if } |u| < 1 \\ 0 & \text{otherwise} \end{cases}$$

3. Epanechnikov (quadratic)

$$D(u) = \begin{cases} 1 - u^2 & \text{if } |u| < 1 \\ 0 & \text{otherwise} \end{cases}$$

4. Biweight (quartic)

$$D(u) = \begin{cases} (1 - u^2)^2 & \text{if } |u| < 1 \\ 0 & \text{otherwise} \end{cases}$$

5. Triweight

$$D(u) = \begin{cases} (1 - u^2)^3 & \text{if } |u| < 1 \\ 0 & \text{otherwise} \end{cases}$$

6. Tricube

$$D(u) = \begin{cases} (1 - |u|^3)^3 & \text{if } |u| < 1 \\ 0 & \text{otherwise} \end{cases}$$

7. Cosine

$$D(u) = \begin{cases} \cos(0.5\pi u) & \text{if } |u| < 1 \\ 0 & \text{otherwise} \end{cases}$$

8. Gaussian

$$D(u) = e^{-0.5u^2}$$

9. Gaussian kernel modified according to (Rikards et al., 2006).

$$D(u, \alpha) = e^{-\alpha u^2}$$

where α is coefficient controlling the locality of the fit (when α is zero, the result of the procedure is equivalent to global regression) and

$$K_\alpha(x_0, x) = D\left(\frac{\|x - x_0\|}{\|x_0 - x_{[n]}\|}, \alpha\right).$$

where $x_{[n]}$ is the farthest of the n data observations to x_0.

Some additional details on the choice of bandwidth (Hastie et al., 2009): Large bandwidth implies lower variance (averages over more observations) but higher bias. Metric window widths tend to keep the bias of the estimate constant, but the variance is inversely proportional to the local density. Nearest neighbors exhibit the opposite behaviour; the variance stays constant and the absolute bias varies inversely with local density. On boundaries, the metric neighborhoods tend to contain less observations, while the nearest-neighborhoods get wider.

Note that the behaviour of the modified Gaussian kernel is inverted – for larger α the fitting is more local.

2.3. Function `lwpeval`

Purpose:

Evaluates predictive performance of LWP using one of the criteria.

Call:

```
[evaluation, df1, df2] = lwpeval(Xtr, Ytr, params, crit, Xv, Yv, weights, failSilently, checkArguments)
```

All the input arguments, except the first four, are optional. Empty values are also accepted (the corresponding defaults will be used).

Input:

`Xtr, Ytr` : Training data. See description of function `lwppredict` for details.

`params` : A structure of parameters for LWP. See description of function `lwpparams` for details.

`crit` : Criterion (string):
`'VD'`: Use validation data (`Xv`, `Yv`)
`'CVE'`: Explicit Leave-One-Out Cross-Validation
`'CV'`: Closed-form expression for Leave-One-Out Cross-Validation
`'GCV'`: Generalized Cross-Validation (Craven & Wahba, 1979)
`'AICC1'`: improved version of AIC (Hurvich et al., 1998)
`'AICC'`: approximation of AIC_{C1} (Hurvich et al., 1998)
`'AIC'`: Akaike Information Criterion (Akaike, 1973, 1974)
`'FPE'`: Final Prediction Error (Akaike, 1970)
`'T'`: (Rice, 1984)
`'S'`: (Shibata, 1981)
The study by Hurvich et al. (1998) compared GCV, AIC_{C1}, AIC_C, AIC, and T for nonparametric regression. Overall, AIC_C gave the best results. It was concluded that AIC_{C1}, AIC_C, and T tend to slightly oversmooth, GCV has a tendency to undersmooth, while AIC has a strong tendency to choose the smallest bandwidth available.
Note that if robust fitting or observation weights are used, the only available criteria are `'VD'` and `'CVE'`.

`Xv, Yv` : Validation data for criterion `'VD'`.

`weights` : Observation weights for training data. See description of function `lwppredict` for details.

`failSilently` : See description of function `lwppredict`. Default value = `false`.

`checkArguments` : Whether to check if all the input arguments are provided correctly. Default value = `true`. It can be useful to turn it off when calling this function many times for optimization purposes.

Output:

`val` : The calculated criterion value.

`df1, df2` : Degrees of freedom of the LWP fit: `df1 = trace(L)`, `df2 = trace(L'*L)`. Not available if `crit` is `'VD'` or `'CVE'`.

Remarks:

The criteria in the LWP toolbox is calculated as follows.

Explicit Leave-One-Out Cross-Validation (Mean Squared Error):

$$\mathrm{CVE}=\frac{1}{n}\sum_{i=1}^{n}\left(y_i-\hat{f}_{(-i)}(x_i)\right)^2$$

where y_i is the response value for the ith training data observation, $\hat{f}_{(-i)}(x_i)$ is the estimate of f obtained by omitting the pair $\{x_i, y_i\}$, and n is the number of observations in the training data.

Closed-form expression for Leave-One-Out Cross-Validation:

$$\mathrm{CV}=\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i-\hat{y}_i}{1-l_{ii}}\right)^2$$

where $\hat{y}_i$ is the estimated value of y_i and l_{ii} is the ith diagonal element of smoothing matrix **L**.

Generalized Cross-Validation (Craven & Wahba, 1979):

$$\mathrm{GCV} = \frac{n\,\mathrm{RSS}}{(n - \mathrm{tr}(\mathbf{L}))^2}$$

where RSS is the residual sum of squares and $\mathrm{tr}(\mathbf{L})$ is the trace of **L**.

Akaike Information Criterion (Akaike, 1973, 1974):

$$\mathrm{AIC} = \log(\mathrm{RSS}/n) + 2\,\mathrm{tr}(\mathbf{L})/n$$

$\mathrm{AIC}_{\mathrm{C1}}$, improved version of AIC (Hurvich et al., 1998):

$$\mathrm{AIC}_{\mathrm{C1}} = \log(\mathrm{RSS}/n) + \frac{(\delta_1/\delta_2)(n + \mathrm{tr}(\mathbf{L}'\mathbf{L}))}{\delta_1^2/\delta_2 - 2}$$

where $\delta_1 = \mathrm{tr}(\mathbf{B})$, $\delta_2 = \mathrm{tr}(\mathbf{B}^2)$, $\mathbf{B} = (\mathbf{I} - \mathbf{H})'(\mathbf{I} - \mathbf{H})$, and $\mathbf{I}$ is the identity matrix.

$\mathrm{AIC}_{\mathrm{C}}$, approximation of $\mathrm{AIC}_{\mathrm{C1}}$ (Hurvich et al., 1998):

$$\mathrm{AIC}_{\mathrm{C}} = \log(\mathrm{RSS}/n) + 1 + \frac{2(\mathrm{tr}(\mathbf{L}) + 1)}{n - \mathrm{tr}(\mathbf{L}) - 2}$$

Final Prediction Error (Akaike, 1970):

$$\mathrm{FPE} = \frac{\mathrm{RSS}(1 + \mathrm{tr}(\mathbf{L})/n)}{n(1 - \mathrm{tr}(\mathbf{L})/n)}$$

T (Rice, 1984):

$$\mathrm{T} = \frac{\mathrm{RSS}}{n - 2\,\mathrm{tr}(\mathbf{L})}$$

S (Shibata, 1981):

$$\mathrm{S} = (1/n)\,\mathrm{RSS}(1 + 2\,\mathrm{tr}(\mathbf{L})/n)$$

2.4. Function `lwpfindh`

Purpose:

Finds the "best" bandwidth for a kernel using a simple grid search followed by fine-tuning. Predictive performances are estimated using one of the criteria provided by function `lwpeval`.

Call:

```
[hBest, critBest, results] = lwpfindh(Xtr, Ytr, params, crit, hList, Xv, Yv, weights, finetune, verbose)
```

All the input arguments, except the first four, are optional. Empty values are also accepted (the corresponding defaults will be used).

Input:

`Xtr, Ytr` : Training data. See description of function `lwppredict` for details.

`params` : A structure of parameters for LWP. See description of function `lwpparams`. Parameter `params.h` is ignored, as it is the one parameter to optimize.

`crit` : See description of function lwpeval.

`hList` : A vector of non-negative window size `h` values to try (see description of function `lwpparams` for details about `h`). If `hList` is not supplied, a grid of values is created automatically depending on `params.kernel`. For all kernel types, except `'GAR'`, equidistant values between 0.05 and 1 with step size 0.05 are considered. For kernel type `'GAR'`, the values grow exponentially from 0 to 100*2^10. The result from this search step is (optionally) fine-tuned (see argument `finetune`).

`Xv, Yv` : Validation data for criterion `'VD'`.

`weights` : Observation weights for training data. See description of function `lwppredict` for details.

`finetune` : Whether to fine-tune the result from the grid search. Nearest neighbor window widths are fine-tuned using more fine-grained grid. Metric window widths are fine-tuned using Nelder-Mead simplex direct search. Default value = `true`.

`verbose` : Whether to print the optimization progress to the console. Default value = `true`.

Output:

`hBest` : The best found value for `h`.

`critBest` : Criterion value for `hBest`.

`results` : A matrix with four columns. First column contains all the `h` values considered in the initial grid search. Second column contains the corresponding criterion values. Third and fourth columns contain `df1` and `df2` values from function `lwpeval`. See function `lwpeval` for details.

3. EXAMPLE OF USAGE

We start by creating a dataset using a two-dimensional function with added noise. The training data consists of 121 observations distributed in a regular 11×11 grid.

```
fun = @(X) (30+(5*X(:,1)+5).*sin(5*X(:,1)+5)) .* (4+exp(-(2.5*X(:,2)+2.5).^2));
[gridX1, gridX2] = meshgrid(-1:0.2:1, -1:0.2:1);
X = [reshape(gridX1, numel(gridX1), 1) reshape(gridX2, numel(gridX2), 1)];
rng(1);
Y = fun(X) + 5 * randn(size(X,1), 1);
```

We will fit 2nd degree local polynomial using the Gaussian kernel with metric window size. To find a good value for the bandwidth, we will use function `lwpfindh` with Leave-One-Out Cross-Validation as a criterion.

First we create a structure of parameters using function `lwpparams` and then we call function `lwpfindh` with these parameters. Note that even though `lwpparams` creates a structure that includes also a value for *h*, `lwpfindh` ignores it, as it is the one parameter to optimize. `lwpfindh` performs a simple Grid Search followed by fine-tuning. Alternatively, we could also supply our own list of candidate values using the input argument `hList` but let's try the automatic mode.

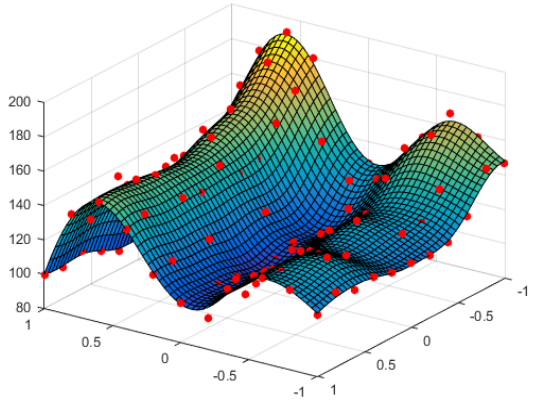
```
params = lwpparams('GAU', 2, false);
[hBest, critBest, results] = lwpfindh(X, Y, params, 'CV');
```

The function prints its progress (you can turn it off by setting the `verbose` argument to `false`) and finally in its output we get the best found *h* (`hBest`), its corresponding criterion value (`critBest`), as well as results from each iteration of the Grid Search (`results`).

Let's update our parameters with the `hBest` value and create a surface plot. For this we will need to predict response values for a range of different inputs. This is done using function `lwppredict`.

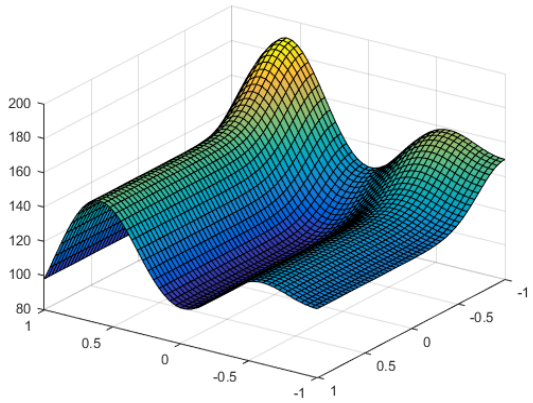
```
params = lwpparams('GAU', 2, false, hBest);

[gridX1, gridX2] = meshgrid(-1:2/50:1, -1:2/50:1);
Xq = [reshape(gridX1, numel(gridX1), 1) reshape(gridX2, numel(gridX2), 1)];
Yq = lwppredict(X, Y, params, Xq);
Yq = reshape(Yq, size(gridX1));
figure;
surf(gridX1, gridX2, Yq);
axis([-1 1 -1 1 80 200]);
hold on;
plot3(X(:,1), X(:,2), Y, 'r.', 'Markersize', 20);
```



For visual comparison, let's plot the true function.

```
Ytrue = fun(Xq);
figure;
surf(gridX1, gridX2, reshape(Ytrue, size(gridX1)));
axis([-1 1 -1 1 80 200]);
```



We can also test our LWP configuration on test data, if we have any. For simplicity's sake, let's just pretend that the grid `Xq` we created for plotting the true function is actually also our test dataset (consisting of 2601 observations). We again use `lwppredict`.

```
MSE = mean((lwppredict(X, Y, params, Xq) - Ytrue) .^ 2)

MSE =
    5.8120
```

Or alternatively, we can also use `lwpeval` with `'VD'` as the criterion:

```
MSE = lwpeval(X, Y, params, 'VD', Xq, Ytrue)

MSE =
    5.8120
```

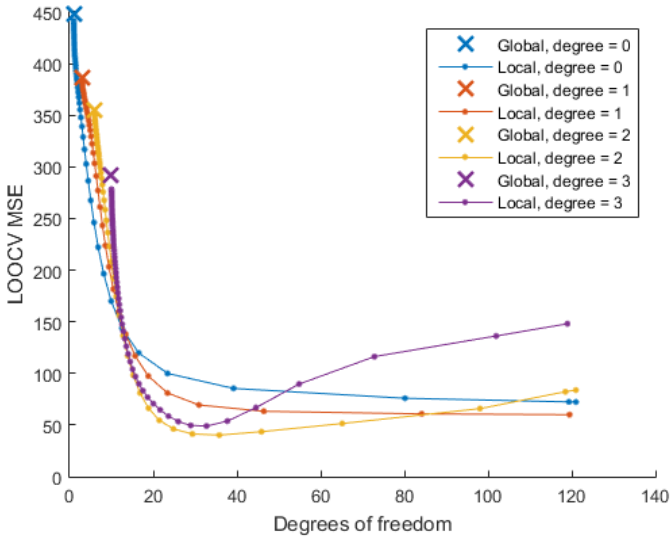
To get a better understanding of how the smoothing parameter *h* influences our results, we can use the output argument `results` of `lwpfindh` for plotting criterion values versus *h* or versus the fitted degrees of freedom of the model (Loader, 1999), similar to the "M plot" proposed by Cleveland & Devlin (1988).

To create the plot, this time we will call `lwpfindh` with our own list of values for *h* (using the input argument `hList`). And while we're at it, let's try all local polynomials of degree 0 through 3. The horizontal axis of the plot will be the fourth column of `results` (fitted degrees of freedom, tr(**L**'**L**)) and the vertical axis will be the second column (criterion value). The fact that for the horizontal axis we are using fitted degrees of freedom, rather than the smoothing parameter *h* (the first column of `results`), aids interpretation and comparability. It allows us to directly compare fits by local polynomials of different degrees (or even different fitting methods).

We will also add global polynomials of the same degrees to the plot. For that we could use some function designed specifically for this purpose but, since we can simulate global polynomials using local polynomials with uniform (rectangular) kernel and bandwidth that spans 100% observations, we will do just that (it is of course much slower than fitting global polynomials directly).

Note that for this plot we don't need the fine-tuning of `hBest`, so we set argument `finetune` of function `lwpfindh` to `false`.

```
figure;
hold all;
colors = get(gca, 'ColorOrder');
for i = 0 : 3
    % Global polynomial
    params = lwpparams('UNI', i, true, 1);
    [MSE, df] = lwpeval(X, Y, params, 'CV');
    plot(df, MSE, 'x', 'MarkerSize', 10, 'LineWidth', 2, 'Color', colors(i+1,:));
    % Local polynomial
    params = lwpparams('GAU', i, false);
    [hBest, critBest, results] = ...
        lwpfindh(X, Y, params, 'CV', 0:0.01:1, [], [], [], false, false);
    plot(results(:,4), results(:,2), '.-', 'MarkerSize', 10, 'Color', colors(i+1,:));
end
legend({'Global, degree = 0' 'Local, degree = 0' ...
        'Global, degree = 1' 'Local, degree = 1' ...
        'Global, degree = 2' 'Local, degree = 2' ...
        'Global, degree = 3' 'Local, degree = 3'}, 'Location', 'NorthEast');
xlabel('Degrees of freedom');
ylabel('LOOCV MSE');
```



We can see that, from the candidates, the 2nd degree local polynomial could indeed be a good choice.

4. REFERENCES

1. Akaike H., Statistical predictor identification, Annals of Institute of Statistical Mathematics, 22 (2), 1970, pp. 202-217.
2. Akaike H., Information theory and an extension of the maximum likelihood principle. Proceedings of 2nd International Symposium on Information Theory, eds Petrov B. N. and Csaki F., 1973, pp. 267-281.
3. Akaike H., A new look at the statistical model identification. IEEE Transactions on Automatic Control, 19 (6), 1974, pp. 716-723.
4. Cleveland W. S., Robust Locally Weighted Regression and Smoothing Scatterplots. Journal of the American Statistical Association, 74 (368), 1979, pp. 829-836.
5. Cleveland W. S., Devlin S. J., Locally weighted regression: An approach to regression analysis by local fitting, Journal of the American Statistical Association, 83 (403), 1988, pp. 596-610.
6. Cleveland W. S., Grosse E., Shyu W. M., Local regression models. Chapter 8 of Statistical Models in S, eds Chambers J. M. and Hastie T. J., Wadsworth & Brooks/Cole, 1992
7. Cleveland W. S., Loader C. L., Smoothing by Local Regression: Principles and Methods, Statistical Theory and Computational Aspects of Smoothing, eds Haerdle W. and Schimek M. G., Springer, New York, 1996, pp. 10-49.
8. Craven P., Wahba G., Smoothing noisy data with spline functions. Numer. Math., 31, 1979, pp. 377-403.
9. Fan J., Gijbels I., Local polynomial modelling and its applications. Chapman & Hall, 1996
10. Hastie T., Tibshirani R., Friedman J., The elements of statistical learning: Data mining, inference and prediction, 2nd edition, Springer, 2009
11. Hurvich C. M., Simonoff J. S., Tsai C.-L., Smoothing Parameter Selection in Nonparametric Regression Using an Improved Akaike Information Criterion, Journal of the Royal Statistical Society, Series B (Statistical Methodology), 60 (2), 1998, pp. 271-293.
12. Loader C., Local Regression and Likelihood, Springer, New York, 1999
13. Nadaraya E. A., On Estimating Regression. Theory of Probability and its Applications, 9 (1), 1964, pp. 141-142.
14. Rice J., Bandwidth choice for nonparametric regression. The Annals of Statistics, 12 (4), 1984, pp. 1215-1230.
15. Rikards R., Abramovich H., Kalnins K., Auzins J., Surrogate modeling in design optimization of stiffened composite shells, Composite Structures, 73 (2), 2006, pp. 244–251.
16. Shibata R., An Optimal Selection of Regression Variables, Biometrika, 68 (1), 1981, pp. 45-54.
17. Stone C., Consistent nonparametric regression. Annals of Statistics, 5 (4), 1977, pp. 595-645.
18. Watson G. S., Smooth regression analysis. Sankhyā: The Indian Journal of Statistics, Series A, 26 (4), 1964, pp. 359-372.