A comparison of polynomials local regressions packages available in R

Alexis Janin, Kien Hoang

I. Introduction

Local regressions[1] is a generalization of the moving average and polynomial regression. It is built on "classical" methods, such as linear and nonlinear least squares regression. They address situations in which the classical procedures do not perform well or cannot be effectively applied without undue labor. Local regressions combine much of the simplicity of linear least squares regression with the flexibility of nonlinear regression. It does this by fitting simple models to localized subsets of the data to build up a function that describes the deterministic part of the variation in the data, point by point.

At each point in the range of the data set a low-degree polynomial is fitted to a subset of the data, with explanatory variable values near the point whose response is being estimated. The polynomial is fitted using weighted least squares, giving more weight to points near the point whose response is being estimated and less weight to points further away. The value of the regression function for the point is then obtained by evaluating the local polynomial using the explanatory variable values for that data point.

The method opens flexible tuning parameters, including the degree of the polynomial model and the weights. The local polynomials fit to each subset of the data are almost always of first or second degree; that is, either locally linear (in the straight line sense) or locally quadratic. The weight function gives the most weight to the data points nearest the point of estimation and the least weight to the data points that are furthest away. The use of the weights is based on the idea that points near each other in the explanatory variable space are more likely to be related to each other in a simple way than points that are further apart. Following this logic, points that are likely to follow the local model best influence the local model parameter estimates the most. There were numerous algorithms explored to finetune this parameter. Most often, weights are given by a bandwidth, representing how far close points should be considered for regression, and a regression Kernel that represents the distribution behavior of the weight assigned.

In this project, a simulation study is performed in order to compare how different packages in R implement the local polynomial regressions, through the data fitting process, and how well it is in data predictions. It is also checked how well the algorithms installed in these packages converge following data increment, as well as their computation times. A simulation study allows for such a consistency check, performed by an increasing to infinity number of datapoints, while it is not possible with real data.

II. Method

1. Global pipeline

Very formally, the problem can be described of the following function that need to be approximated for the different packages:

$$f(h, d, N, g, n, K) \rightarrow (t, a)$$

where h is the bandwidth, d is the degree, N the dataset size, g the function (more or less smooth and regular) that need to be approximated and n the noise of the function g, while K stands for the kernel used. The outcome are t the time needed for the computation of the package 'f', and a his accuracy (that will be simply considered as a squared L^2 norm, so simply the sum of squares of the error).

This is in practice completely impossible to do, but this helps to clarify what is the goal and which direction this paper does explore or not.

2. Parameters description

i) Sampling size

The data for this study are simulated from a fixed input. Our input, X, are sampled from a fixed range from 0 to 5, with equal spaces between the points. The numbers of samples N for training are set to be 50, 100, 500, 1000, and 4500, while evaluation dataset are fixed to be 1000.

ii) Functions estimated

The outputs Y are computed based on multiple functions, so that the algorithms; behaviors following each function are observed. The functions chosen are, ranked from the more to the less smooth:

- Mixed normal distribution function: the function is the sum of probability density distribution with the means equalling 1, 3 and standard deviations equalling 1, 0.5 correspondingly. The contribution proportions are set as 0.7 and 0.3 correspondingly.

  Overall formula:

$$Y = f_1(X) = 0.7 * N(X, 1, 1) + 0.3 * N(X, 3, 0.5)$$

- Log and square root: the function is a very smooth function. .

  Overall formula:

$$Y = f_2(X) = log(1/(1 + X)) * \sqrt{X}$$

- Log sin: the function is a non-smooth function; however, its smoothness can be controlled using its parameters.

  Overall formula:

$$Y = f_3(X) = log(sin(2.5 * X) + 1.05)$$

- Weierstrass[2]: the Weierstrass function is an example of a real-valued function that is continuous everywhere but differentiable nowhere.

  Overall formula:

$$Y = f_4(X) = \sum_{n=0}^{\infty} 0.9^n * cos(9^n * X * \pi)$$

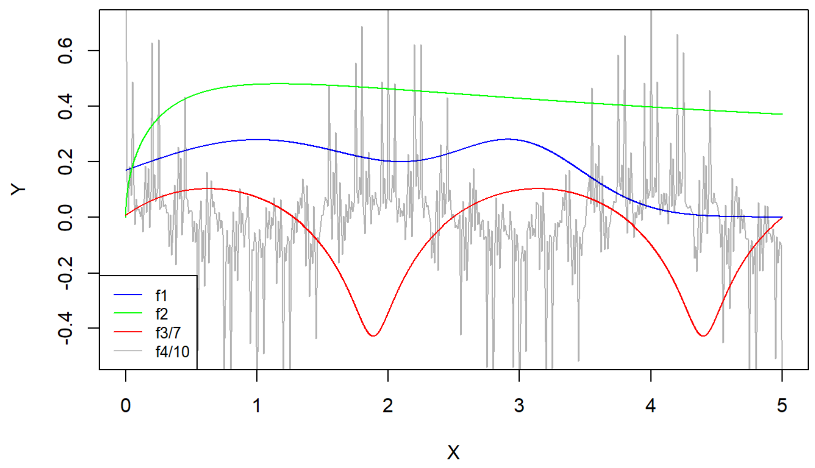


Figure 1: functions (without noise) used to compare the methods, adjusted to be comparable on the Y axis

iii) Noise induction

In addition, we inject some noise to the training data output, which are from the normal distributions, to confuse the model. The evaluations are not injected, to best evaluate how the algorithms learn the underlying function. The variance of the noise is chosen with regard to the total variation of the function, in order to give a possibility to packages to indeed recover the initial behavior of the function. We recall that $TV(f) = max_x f(x) - min_x f(x)$.

The function of the noise is assumed to be from normal distribution, with mean 0, and the variance of the noise is chosen as above. The noise will be injected to the output after its calculation.

iv) Other parameters

The Kernel choice could impact the different models in different ways, however for the sake of clarity we chose to not vary this parameter, since it is often much less important than the bandwidth choice, and a lot of parameters are already compared.

The bandwidth and degree are not fixed but will vary if different ranges to compare the methods, with reasonable choice for the 5 X-length of our dataset. The degrees are picked from

1 to 7, which has already included the popular default choice of 2 for most packages. The bandwidths are more tricky; we decided to range the bandwidths selection from 0.15 to 0.55, with 0.1 different. The bandwidths are small enough to cover a small portion of the data, and can rise up to 10% of the data for global considerations.

For the error and time computation, each simulation has been runned 6 times and averaged. Since the error and the time should be very accurate, 6 times should be enough to avoid any pathological random effect coming from the seed. When a noisy input is made, it is then used for all the methods, reducing even more the potential random effects between different packages.

3. Packages

We explore the following packages that implemented the local regression in R:

i) locpoly [3]

The package contains local polynomial fitting with a kernel weight, which is used to estimate either a density, regression function or their derivatives. In the case of density estimation, the data are binned and the local fitting procedure is applied to the bin counts. In either case, binned approximations over an equally-spaced grid are used for fast computation. The bandwidth may be either scalar or a vector of length *gridsize*.

ii) locpol [4]

The package provides a basic interface like the *lm* functionality. *summary* and *print* methods show very basic information about the fit, *fitted* return the estimation of the derivatives if deg is larger than 0, and *plot* provides a plot of data, local polynomial estimation and the variance estimation.

iii) locfit [5]

locfit is the model formula-based interface to the Locfit library for fitting local regression and likelihood models. *locfit* is implemented as a front-end to *locfit.raw*. However, the bandwidth method applied in this method is different from other methods, requiring appropriate substitution (occur in the codes). The bandwidth here is relative to a percentage of the dataset that should be considered for each of the data point regression.

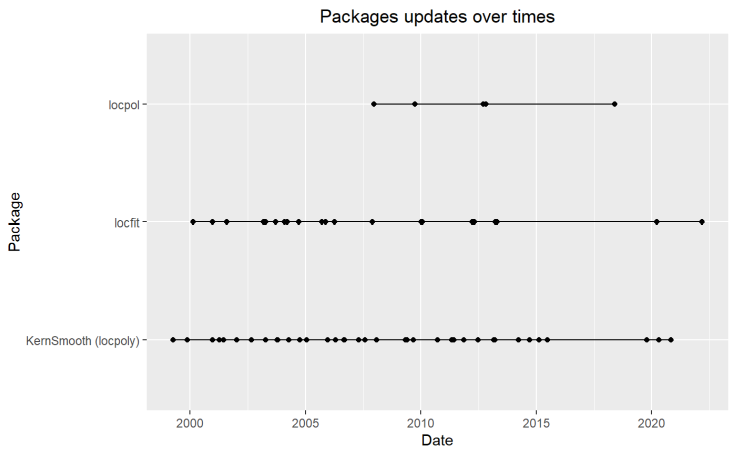


Figure 2 : Update time comparison of each packages considered

In the figure 2, one can see that between the 3 packages that will be compared in the next part, all of them are regularly updated lastly, even if *locpol* has much fewer updates and is more recent than the others.

In addition, there are other packages such as *lpridge(lpepa)*, *stats(loess/lowess)*, *lokern(lokerns)* but they miss some important features in order to be well compared with everything. As example *lpepa*[6] stands for lp_epachnenikov and then is not using a gaussian kernel; *loess* has no bandwidth argument but rather a span one which has a totally different way of working.

3. Comparison pipeline

We will measure not just the accuracy, but also the consistency of the packages with respect to each other and themselves, for each case of the parameters, which are the bandwidth and the degree of the polynomials. We iterate through the parameters, with bandwidths h ranging from 0.15 to 0.55, with 0.1 difference, while the degree of the polynomials ranging from 1 to 7. We expect in each case, the packages returning values close to each other.

In addition, we measure the computation time t, which is defined as the time needed for the package to produce his results, which is sometimes directly the prediction points used for the above error calculation, sometimes parameters to estimate them etc… packages can be very different in this aspect and we do not explore this part either.

Finally, we also iterate through the number of points used for training in each algorithm. We expect as the number of training data increases, the error of estimating new data will decrease. The error considered will be an squared L^2error as said above, consisting of the difference between the predicted 1000 points linearly spaced between 0 and 5, and the true 1000 points that the original function (without noise) would have produced.

<u>III. Result Analysis</u>

For better visualizations and analysis, some parameters will be fixed while others will be visualized.

1. Time comparison

Figure 3 shows the typical behavior of the packages regardless of the function: the *locpol* package is much slower than the two others, and this gets even worse when the dataset size increases. *locpol* appears to increase the computation time linearly with data size, since the space between the 50, 500, and 4500 input space size is the same, over a log-scale axis. However the other packages remain very fast in time computation. In the figures, the bandwidth appears to have no influence on the computation time, and the degree has a very small impact.

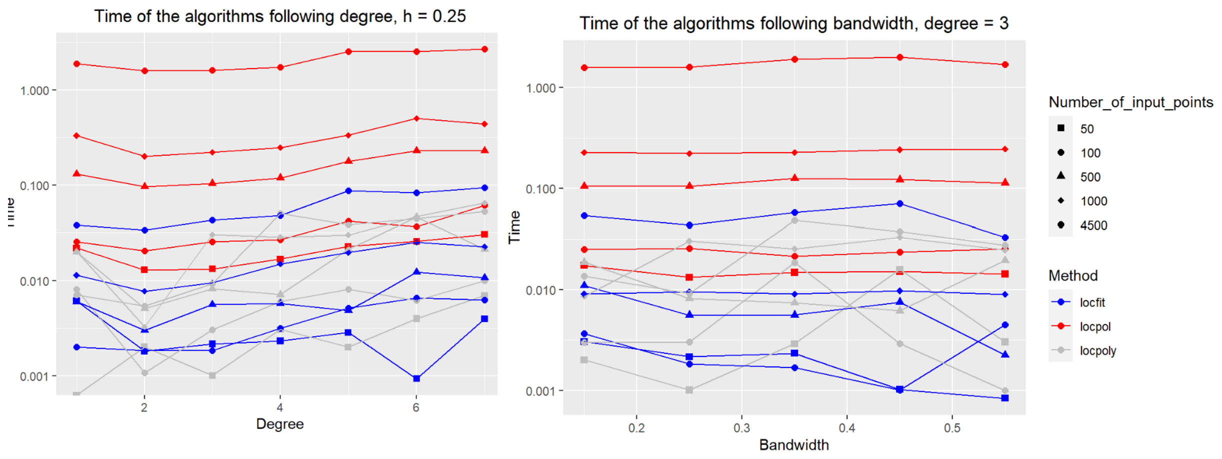


Figure 3: Comparison of algorithms' running times, with respect to bandwidth and degree

The result of time comparison is similar to what we have expected, as the substitution of the bandwidth only leads to different substitution of numbers in the kernel matrix calculations, while the increment of degree does change the dimensions of the matrices, which slightly increases the computation time. For more details, please have a look at the Appendix for more discrete visualizations.

2. Error analysis

The error comparison is much more tricky, and changes following all parameters. Below are the most different behaviors one could extract from our framework**.** For the figure 4, the ground-truth function is f_3: the logsin function described in the method part. It is smooth enough so with the right parameters and method the approximation is good; but not for all of them. An error below 1 is a very good approximation, while from 100 to 1000 it is not very satisfying. To get to that, the noise was chosen gaussian with $\sigma = \frac{TV(f_3)}{35}$.

For example, one can see that the methods *locpoly* and *locpol* got nearly the same error behavior for most of the functions, with a significant shift between them, as the evaluated errors are calculated on a log scale.

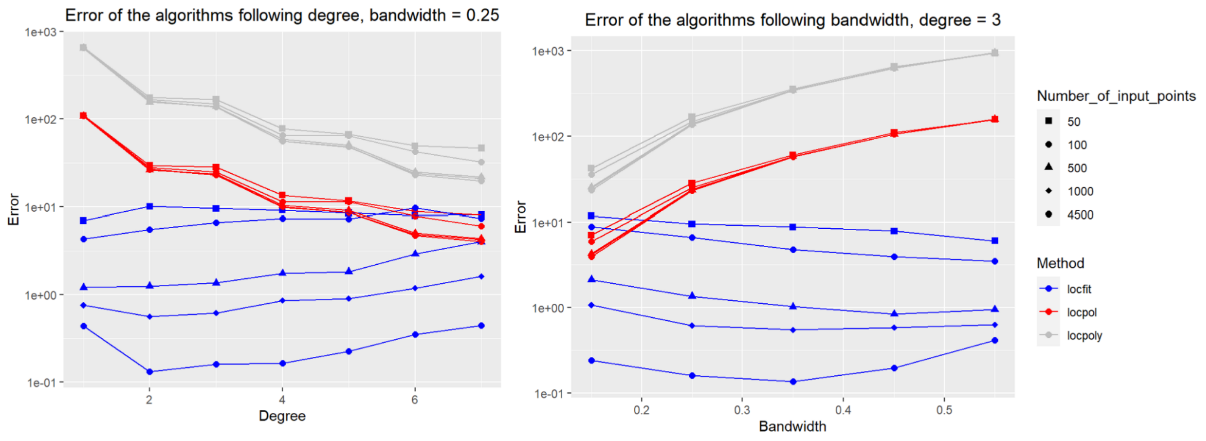


Figure 4: Errors for evaluating logsin (f_3) with respect to bandwidths and degree

The figure 4 was for "high" errors, the figure 5 below shows behavior of the methods for a smoother function (with $\sigma = TV(f_2)/10$ as noise standard deviation), and then the errors are then much lower, from 0.01 to 3 for the highest. One can then observe the very different behaviors: the *locpol* method seems to work better than the other methods in this case, even if the *locfit* one with enough points and the right parameters is still performing very well. One can also see that with more points the error is still decreasing even when it is relatively low. With regard to how regular f_2 is, the error nearly only comes from the noise and not the function itself, and so figure 4 suggests that for purely denoising purposes, with a lot of data points and not too much noise, *locpol* could be the best package.

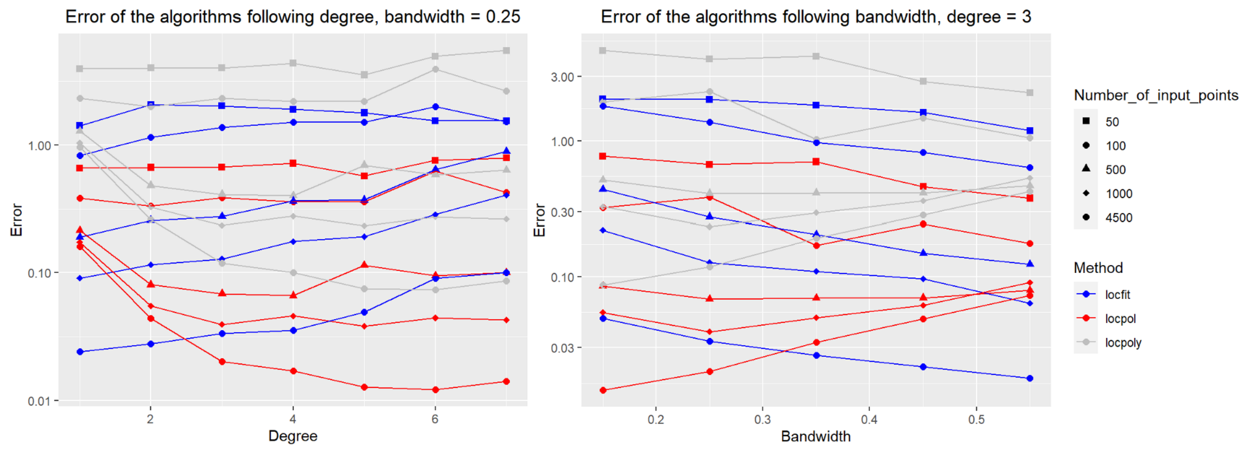


Figure 5: Errors for evaluating f_2 with respect to bandwidths and degree

3. Convergence Analysis

Overall, the error decreases significantly, when we add more points to the training data. The most significant drops can be observed when we increase the sample size from 100 to 1000. We show the errors' fluctuations with bandwidth as the x-axis and error for the y-axis, as the number of bandwidths. This can blur the visual effects of how sample sizes affect the errors, but allow us to look at less distinct lines, and allow better analysis. The function we showed in this part is the Weierstrass function (f_4), but all of the functions can be found in the Appendix.

i) locfit

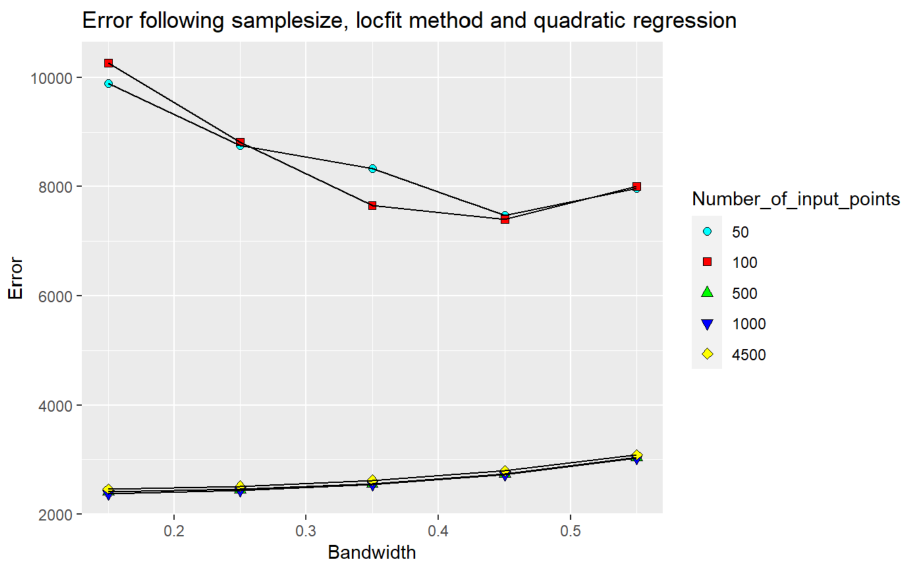


Figure 6: Errors for evaluating f_4 at different levels of number of samples for *locfit*

The error of computations drop significantly when we increase the number of samples from 100 to 500. The errors between 500 and 4500 do not vary much during all bandwidths, showing a sufficient amount of data at 500 samples. An additional point in this case is that the change of number of samples from 100 to 500 at bandwidth equalling 0.15 shows the most significant change (from largest errors to smallest errors).

ii) locpol

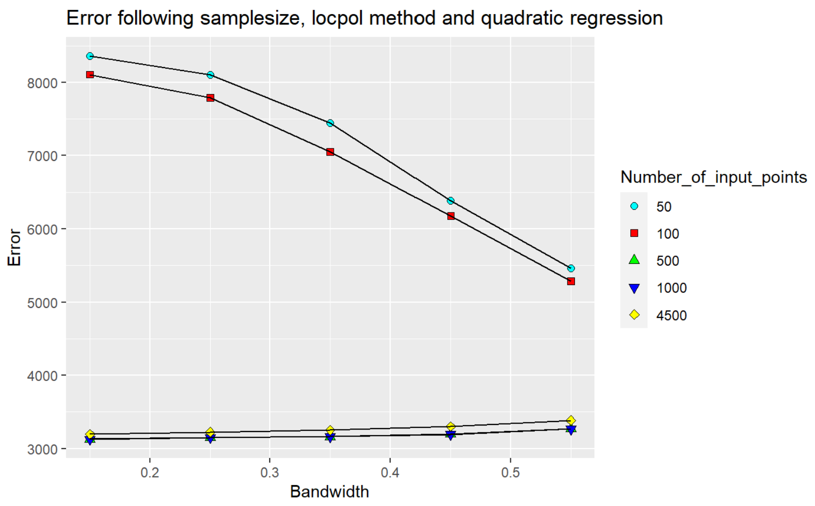


Figure 7: Errors for evaluating f_4 at different levels of number of samples for *locpol*

The errors also support that increasing the number of samples from 100 to 500 most significantly affects the local regressions' efficiency. We also acknowledge that the number of samples at 4500 output worse results than 1000, which is consistent with the *locfit* package. Yet, in the last case, where bandwidth equalling 0.55, increasing sample sizes have a much smaller effect than the other bandwidths, which may be due to the bandwidths themselves.

iii) locpoly

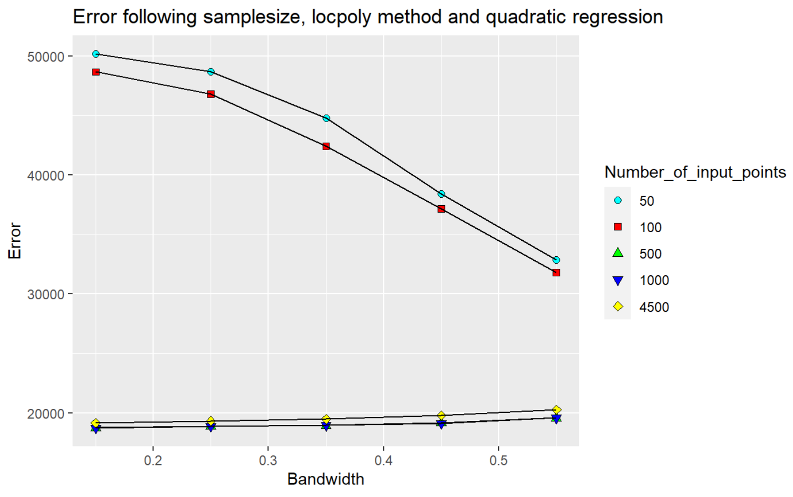


Figure 8: Errors for evaluating f_4 at different levels of number of samples for *locpoly*

Similar to *locpol*, the errors do reduce when the data are increased in size. We also observe less and less reductions as the bandwidth increases, similar to *locpol*.

IV Discussion and conclusion

1. Conclusion

One could observe from the results that the function choice doesn't affect the different computation time between the methods, and doesn't affect either the ranking of the errors between the methods (see Appendix). Therefore since locfit and locpoly are faster than the *locpol* method, and that locfit is more accurate than *locpoly* and *locpol*, it is then advised to first think about *locfit* as a local polynomial regression method. Moreover the bandwidth choice is easier since it does not depend on the data range. One drawback of this method are the very low bandwidths which cannot be taken.

The *locpol* method seems like the worst method but is also the one that has been the least recently updated, and the most recent one. One could wonder that might change with more updates and development. Despite everything, *locpol* still seems to perform well in denoising functions under particular conditions, and then shouldn't be forgotten in all cases.

All the methods seem to improve when adding new data, but only up to a point. This is an interesting perspective, as it is on the same page with the problems of underfitting/overfitting the models, 'more data does not guarantee better results'.

As often simulation studies allow for a lot of flexibility and the use of the true functions since they are known by design. This is especially useful in cases where a lot of parameters are involved and the methods behavior's is difficult to understand theoretically.

2. Discussion

The main limitation of this project is clearly the lack of global generality of the comparison. It might depend on future updates of the packages, precise cases etc… All packages have their own limitations, and this project restricts itself to these limitations for all packages. For a big dataset one shouldn't use *locpol* since it requires less than 5000 data points, but for easier visualization one should use *locpol*, since the plot associated function allows to see directly the confidence intervals, fitting curve etc… *locfit* shouldn't be used for a too narrow bandwidth, which could be useful for a big dataset very noisy and not smooth at all.

Moreover, we explored only gaussian noise, where other types of noise could be interesting such as exponential or uniform ones. We used only linearly spaced points over the x axis, but this is not always the case in the real world, neither a standard noise with the same variance for all x. These are limitations of our framework. It is very difficult to present nicely a lot of parameters changing, coming from the extremely varied cases that the local polynomial regression methods aim to estimate.

The packages should bring very close results from their documentation and the adjustments that were done. This is the case between *locpol* and *locpoly*, which behave the same way and are indeed apparently implemented in the same way (no adjustment needed), but it still has some differences following the functions or parameters. The *locfit* method seems to not be implemented in the same way even with the adjustment. Indeed, an issue of this framework is that it’s hard to make methods comparable (then should give the same results with the same parameters) when the goal is precisely to find the differences.

Also, if one needs regression but with different hypotheses than the one stated here, it might be useful to consider all the packages presented in [3. packages] and if no polynomial regression is needed, the comparison from [7] might be useful to obtain information and local regression of order 1.

References

[1] Cleveland, William S. (1979). "Robust Locally Weighted Regression and Smoothing Scatterplots". Journal of the American Statistical Association. 74 (368): 829–836. doi:10.2307/2286407. JSTOR 2286407. MR 0556476.

[2] Weierstrass, Karl (18 July 1872), Über continuirliche Functionen eines reellen Arguments, die für keinen Werth des letzeren einen bestimmten Differentialquotienten besitzen, Königlich Preussische Akademie der Wissenschaften

[3] Wand, M. P. and Jones, M. C. (1995). Kernel Smoothing. Chapman and Hall, London.

[4] Fan, J. and Gijbels, I. Local polynomial modelling and its applications\/. Chapman & Hall, London (1996).

[5] Loader, C. (1999). Local Regression and Likelihood. Springer, New York

[6] Seifert, B. and Gasser, T. (1996) Finite sample variance of local polynomials: Analysis and solutions. J. American Statistical Association 91(433), 267–275.

[7] Deng, H. and Wickham, H. (2011) Density estimation in R.

Seifert, B. and Gasser, T. (2000) Data adaptive ridging in local polynomial regression. J. Computational and Graphical Statistics 9, 338–360.

Seifert, B. and Gasser, T. (1998) Ridging Methods in Local Polynomial Regression. in: S. Weisberg (ed), Dimension Reduction, Computational Complexity, and Information, Vol.30 of Computing

Science \& Statistics, Interface Foundation of North America, 467–476.

Seifert, B. and Gasser, T. (1998) Local polynomial smoothing. in: Encyclopedia of Statistical Sciences, Update Vol.2, Wiley, 367–372.

Seifert, B., and Gasser, T. (1996) Variance properties of local polynomials and ensuing modifications. in: Statistical Theory and Computational Aspects of Smoothing, W. Härdle, M. G. Schimek(eds), Physica, 50–127

APPENDIX

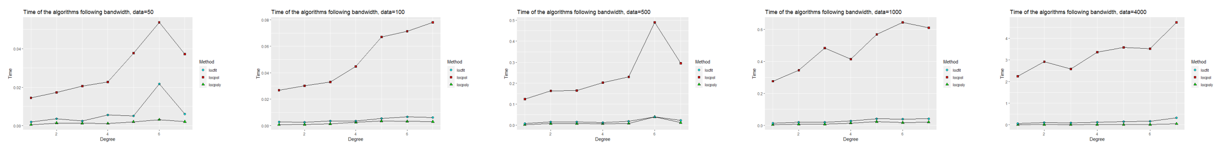
1. Time comparison

In this subsection, we suppose that the bandwidths calculations are related to substitution of numbers, and do not significantly change the time. Thus, we fix the bandwidths, and consider the change of data, as well as the changes of the degree of polynomials, which are similar to adding more coefficients. (we show up to 5 for better visualizations)

Overall, the running time increases for all packages when we increase the degrees of the polynomials. The trend seems to be linear in most cases, especially for the packages locfit and locpoly at high degrees.

Observably, the running time of *locpol* is the greatest in all cases, while *locpoly* is the fastest algorithm. However, *locfit* does keep a closed record to the *locpoly*.

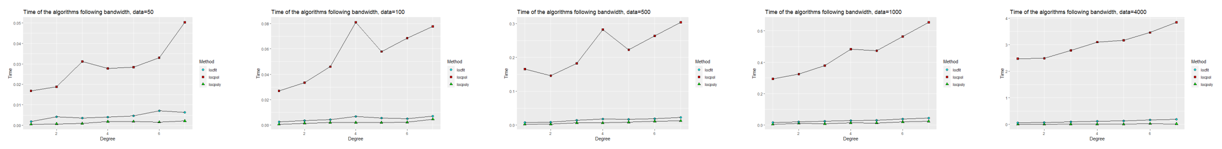
i) $Y = 0.7 * N(X, 0, 1) + 0.3 * N(X, 2, 3)$



Supplement figure 1: Running time of the algorithms following the degrees computing f_1

Except for when the degree is 6, which makes the running time for the packages rise substantially, there is a steady rising trend of running time when the degree increases.

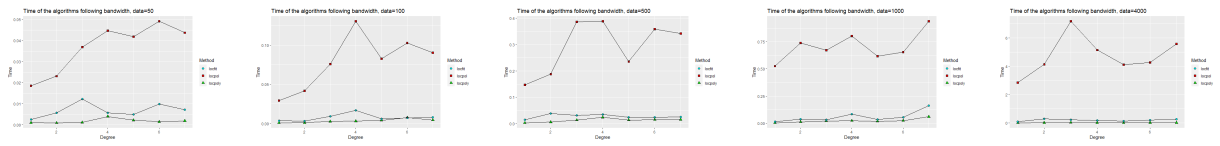
ii) $Y = log(sin(5 * X) + 1.05)$



Supplement figure 2: Running time of the algorithms following the degrees computing f_2

The time still shows increasing trends, yet, peaks at different degrees for the packages.

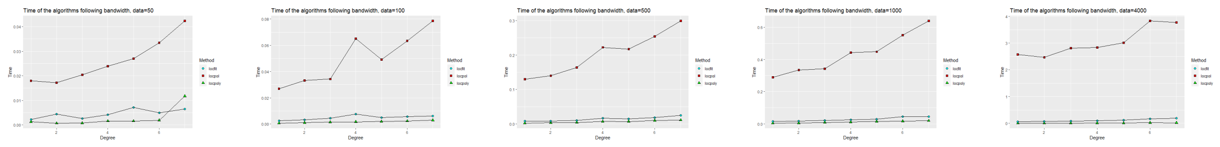
iii) $Y = log(1/(1 + X)) * \sqrt{X}$



Supplement figure 3: Running time of the algorithms following the degrees computing f_3

The *locpol* and *locfit* packages show high fluctuations at different degrees, which is unexplainable. *locpoly* is still consistent, except at degree being 1.

iv) $Y = \sum_{n=0}^{\infty} 0.9^n * cos(9^n * X * \pi)$



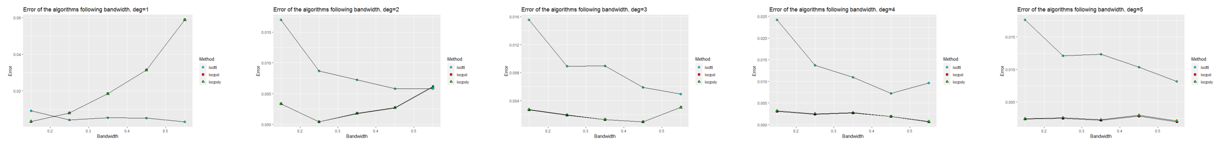
Supplement figure 4: Running time of the algorithms following the degrees computing f_4

Stable trends (except for degree equal to 2) for *locpol*; yet, *locfit* still shows high fluctuations at small degrees.

2. Error comparison of different bandwidths

In this section, we fix the number of datapoints used to the maximum (4500). The results for each function are plotted below. It is observable that the *locpol* and *locpoly* highly agree with each other in terms of errors; yet, *locfit* frequently produces lower errors.

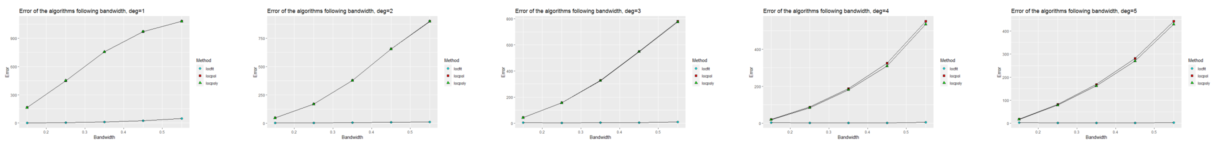
i) $Y = 0.7 * N(X, 0, 1) + 0.3 * N(X, 2, 3)$



Supplement figure 5: Error of the algorithms following the bandwidths computing f_1

It is strange that for large bandwidths, the packages seem to agree at degree 2 and 3. For higher degrees, *locfit* shows much smaller errors. For degree equalling 1, the packages show contradictory behaviors, as *locfit* reduces in errors as bandwidths increase, while the other two increases significantly.

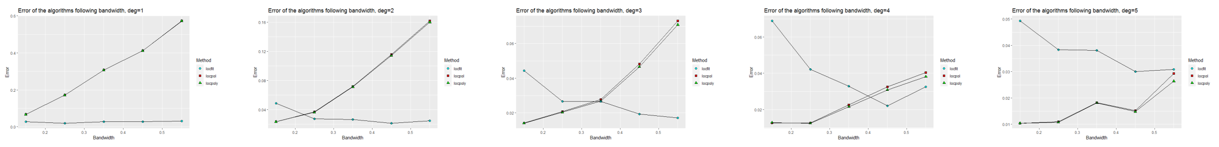
ii) $Y = log(sin(5 * X) + 1.05)$



Supplement figure 6: Error of the algorithms following the bandwidths computing f_2

locfit rarely changes in terms of errors, while two other packages increase their errors as bandwidths increase.

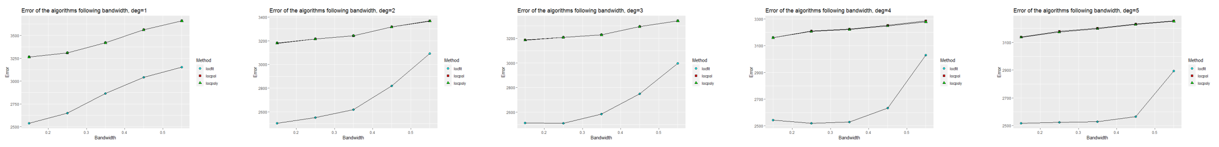
iii) $Y = log(1/(1 + X)) * \sqrt{X}$



Supplement figure 7: Error of the algorithms following the bandwidths computing f_3

It is noteworthy that for small degrees, errors of *locfit* reduces compared to the increment of the other two packages. *locfit* starts at higher errors than the other two, while ends up at much lower errors. However, for high degrees, the other two packages perform better over *locfit*. They seem to converge and agree on the errors at high bandwidths.

iv) $Y = \sum_{n=0}^{\infty} 0.9^n * cos(9^n * X * \pi)$



Supplement figure 8: Error of the algorithms following the bandwidths computing f_4

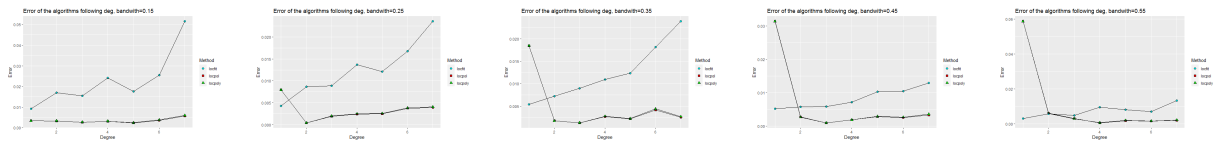
locfit once again shows lower error. However, all three packages share the patterns of increasing errors at larger bandwidths.

3. Error comparison of different degrees

The number of datapoints are also fixed to the maximum (4500).

It is observable that the *locpol* and *locpoly* highly agree with each other in terms of errors; yet, *locfit* frequently produces lower errors.

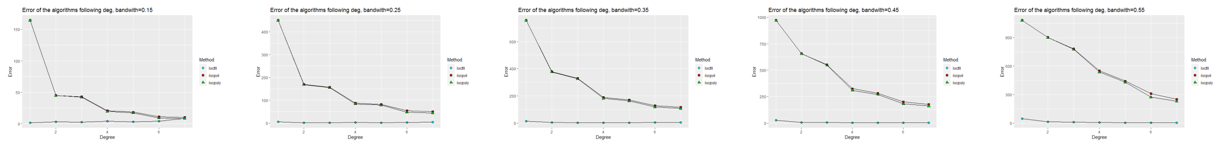
i) $Y = 0.7 * N(X, 0, 1) + 0.3 * N(X, 2, 3)$



Supplement figure 9: Error of the algorithms following the degrees computing f_1

locfit seems to have the behavior of overfitting, with more data coming up with more errors. However, the increasing degrees does help the other two packages to learn the function better, especially from 1 to 2 (this is the default mode for most packages).

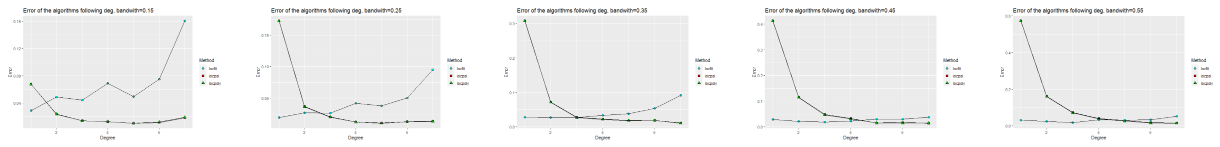
ii) $Y = log(sin(5 * X) + 1.05)$



Supplement figure 10: Error of the algorithms following the degrees computing f_2

The same situation occurs, as *locfit* sees no improvements, while the other two packages show improvements quadratically or linearly.

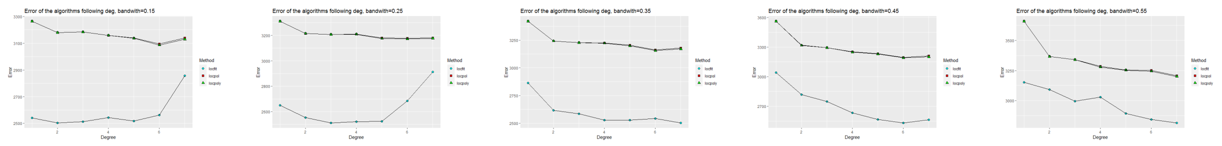
iii) $Y = log(1/(1 + X)) * \sqrt{X}$



Supplement figure 11: Error of the algorithms following the degrees computing f_3

The same situation occurs, as *locfit* sees no improvements, and even worse performance, while the other two packages show improvements quadratically or linearly. The two packages performed much better than *locfit* at high bandwidths, at any degrees.

iv) $Y = \sum_{n=0}^{\infty} 0.9^n * cos(9^n * X * \pi)$



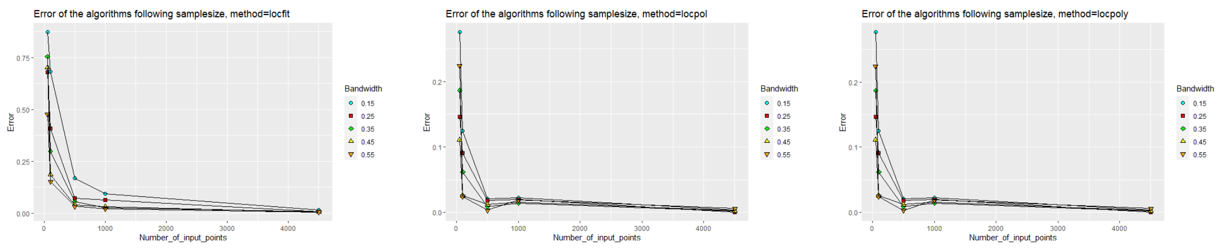
Supplement figure 12: Error of the algorithms following the degrees computing f_4

locfit performs better than the other packages, yet it once again showed overfitting behaviors at more local bandwidths, which can be explainable by the fact of higher dims at more local interpretations. The other two packages show consistent decrement.

4. Convergence comparison

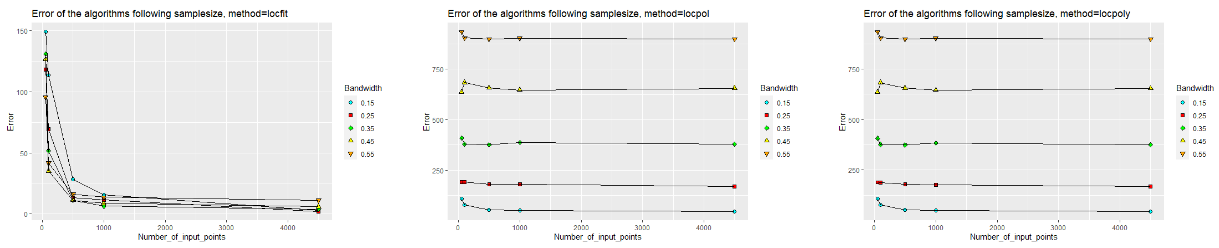
In this analysis, we will fix the degree of the polynomial. We will draw the sample sizes as the levels to compare. We all saw that (except for f_2) the error reduces significantly for all bandwidths and packages. Yet, the best number of datapoints is frequently 500 or 1000, showing that more data does not guarantee better performance. We comment on special case of f_2

i) $Y = 0.7 * N(X, 0, 1) + 0.3 * N(X, 2, 3)$



Supplement figure 13: Error of the algorithms at different sample sizes computing f_1

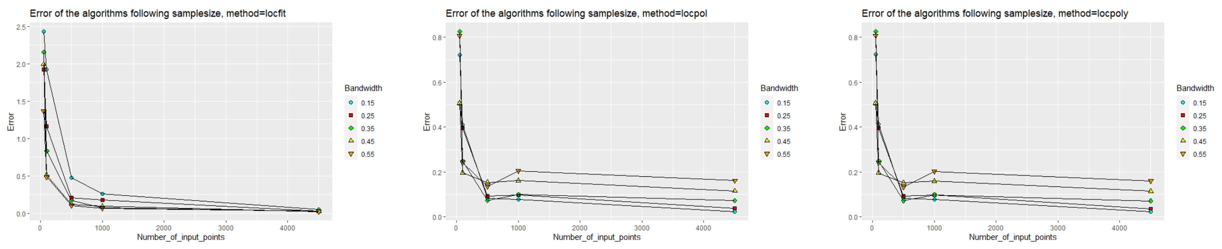
ii) $Y = log(sin(5 * X) + 1.05)$



Supplement figure 14: Error of the algorithms at different sample sizes computing f_2

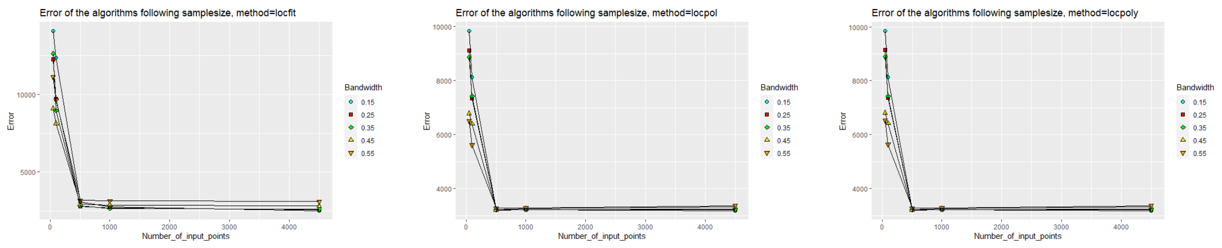
locfit shows the mentioned behaviors, of reductions in errors with more data. However, *locpol* and *locpoly* cannot learn well in this case, with no reductions in errors at all.

iii) $Y = log(1/(1 + X)) * \sqrt{X}$



Supplement figure 15: Error of the algorithms at different sample sizes computing f_3

iv) $Y = \sum_{n=0}^{\infty} 0.9^n * cos(9^n * X * \pi)$



Supplement figure 16: Error of the algorithms at different sample sizes computing f_4

Mentioned in the main analysis.