Loess.jl

Loess.jl is a domain-specific package for performing local regression, which is a technique for fitting a smooth curve to data points that accounts for local trends or patterns in the data. Loess stands for "locally estimated scatterplot smoothing," and the technique involves fitting a separate regression line to each data point based on nearby data points.

The Loess.jl package provides an implementation of the loess algorithm in Julia, which is a popular method for performing local regression. The package includes functions for fitting loess models to data, as well as tools for visualizing the results.

The JuliaStats organization on GitHub, which maintains this package, is dedicated to developing packages for statistical computing and data analysis in the Julia programming language.

Stakeholders

Who develops the software and who pays for it?

Loess.jl is an open-source software package that is developed and maintained by contributors from the Julia community, primarily by the JuliaStats organization. The package is freely available to anyone who wishes to use it, and there is no single organization or entity that pays for its development.

Who uses it?

The users of Loess.jl are typically individuals and organizations who work in the field of statistics and data analysis, particularly those who are interested in performing local regression on their data. This could include researchers, academics, data scientists, and analysts across a wide range of industries, including finance, healthcare, and engineering.

How do they communicate and collaborate?

Communication and collaboration within the Julia community generally takes place through online forums, such as the Julia Discourse forum or GitHub, where developers can discuss issues, share ideas, and work together on code. The developers of Loess.jl are also open to feedback and contributions from the broader community, and encourage users to report issues and suggest improvements.

Who is impacted (positively or negatively) by use of the software?

Anyone who works with data and is interested in local regression could potentially use Loess.jl, regardless of their industry or field of study. However, the impact of the software on different users will depend on their specific use case. For some users, Loess.jl may be a valuable tool that enables them to analyze and understand their data more effectively. For others, it may have little or no impact on their work.

Overall, the impact of Loess.jl on users and stakeholders is likely to be positive, as it provides a powerful and efficient tool for performing local regression in Julia, which can help to improve the accuracy and insight of data analysis. However, there may be some users who are negatively impacted by the software if they encounter bugs or issues that prevent them from using it effectively.

Metrics and features

How do concepts like accuracy, conditioning, stability, and cost appear?

Accuracy:

In the context of Loess.jl, accuracy refers to how well the loess model fits the data. A loess model with high accuracy will have a low sum of squared residuals, indicating that the model captures the underlying structure of the data well. Accuracy can be evaluated using metrics such as the mean squared error or the coefficient of determination (R-squared).

Conditioning:

Conditioning refers to how well-conditioned the design matrix is when fitting the loess model. In general, a well-conditioned matrix has a small condition number, which indicates that small changes in the input data will result in small changes in the output of the loess model. A poorly-conditioned/ill-conditioned matrix has a large condition number, which can lead to numerical instability and inaccuracies in the fitted model. Loess.jl uses a modified QR decomposition to handle conditioning issues and ensure numerical stability.

Stability:

Stability refers to how robust the loess model is to small perturbations in the input data. A stable model will produce similar results even if the input data is slightly changed, while an unstable model may produce vastly different results for small changes in the input. Loess.jl is designed to be stable by default, with the span parameter controlling the degree of smoothing applied to the data to prevent overfitting.

Cost:

Cost refers to the computational cost of fitting the loess model to the data. Loess.jl is designed to be computationally efficient and can handle large datasets with millions of observations. However, the cost of fitting a loess model can still be significant for very large datasets, especially if high accuracy or conditioning is required. In these cases, it may be necessary to use specialized algorithms or hardware to speed up the computation.

If the software is fast, how is "fast" defined?

In the case of the Loess.jl package, the definition of "fast" is likely related to the computational efficiency of the loess algorithm used to fit smooth curves to data.

The efficiency of the loess algorithm can be measured in terms of its computational complexity, which is the amount of time and memory required to perform the algorithm as a function of the input size. For example, a linear-time algorithm (O(n)) will scale linearly with the size of the input data, while a quadratic-time algorithm (O(n^2)) will scale quadratically with the size of the input data.

In general, a "fast" loess algorithm should have a computational complexity that is either linear or near-linear in the size of the input data. The Loess.jl package is designed to be fast by using an algorithm that has a near-linear complexity with respect to the size of the input data. Specifically, the loess algorithm used in Loess.jl has a complexity of O(n log n) or O(n) depending on the chosen values of the span and degree parameters.

In addition to computational complexity, other factors that can impact the speed of software include the efficiency of the underlying programming language, the use of specialized hardware (e.g., GPUs), and the optimization of the code for parallel processing or distributed computing.

Are there modeling decisions made in the interest of good conditioning? Are there algorithmic choices made for stability?

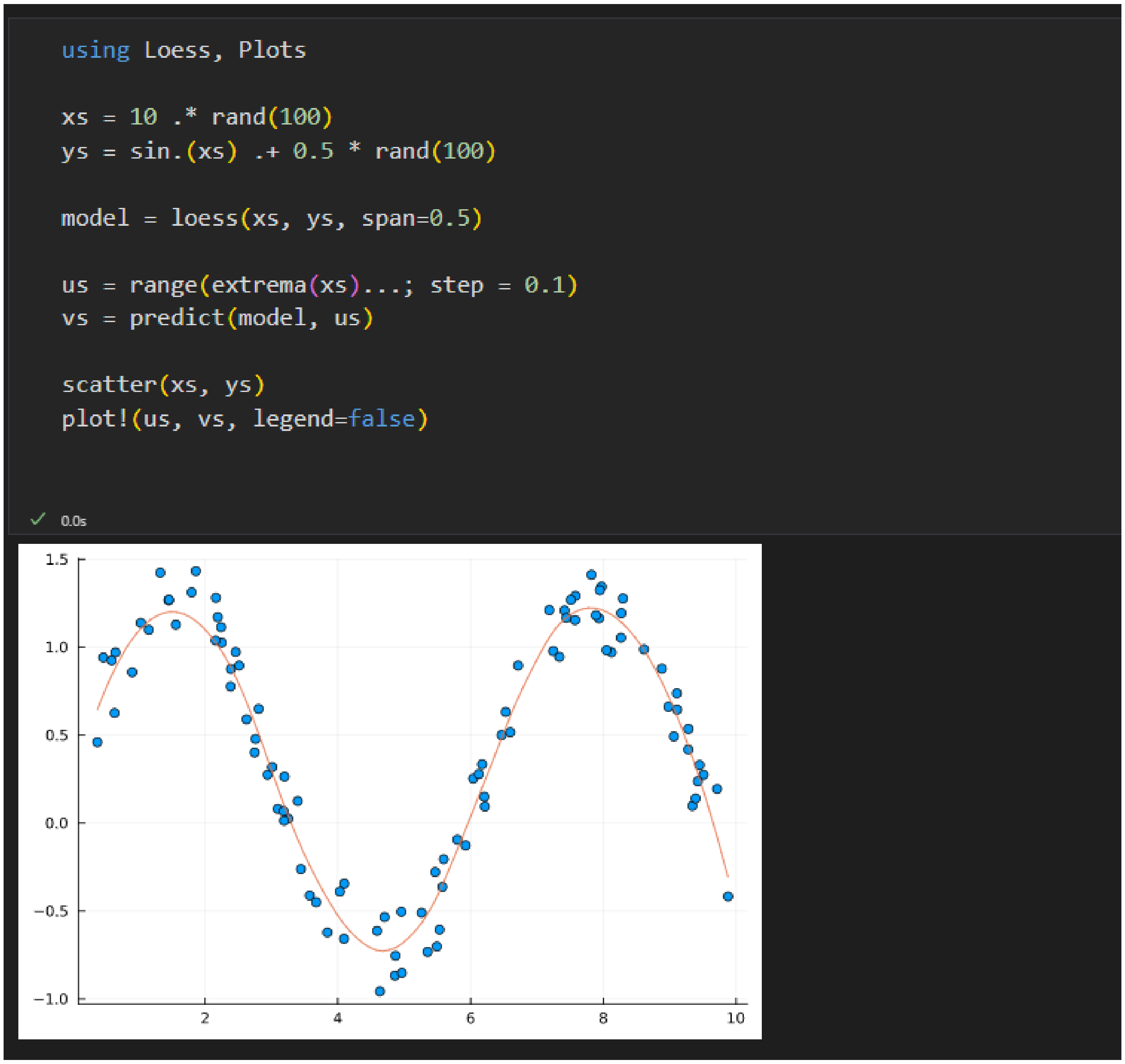
Yes, the Loess.jl package makes modeling decisions that are intended to promote good conditioning and numerical stability when fitting loess models.

In particular, the package includes several options for controlling the smoothing parameter span and the degree of the polynomial fit, which can affect the conditioning of the underlying linear system and the stability of the fitted model. By default, the package uses a "robust" method for estimating the smoothing parameter, which is designed to be less sensitive to outliers and extreme values in the data. This can help improve the conditioning and stability of the fitted model in cases where the data may contain noise or other sources of variability.

Additionally, the package includes options for normalizing the input data and for specifying the bandwidth of the loess smoother, which can help improve the conditioning and stability of the algorithm. The Loess.jl package also uses efficient algorithms and data structures (such as kd-trees and sparse matrices) to speed up the computation and reduce memory usage, which can help avoid numerical issues that can arise with large or complex datasets.

Overall, the Loess.jl package is designed to balance the trade-off between computational efficiency, accuracy, and numerical stability when fitting loess models, and includes several modeling decisions and algorithmic choices that are intended to promote good conditioning and stability.

Running loess locally.



```
using Loess, Plots

xs = 10 .* rand(100)
ys = sin.(xs) .+ 0.5 * rand(100)

model = loess(xs, ys, span=0.5)

us = range(extrema(xs)...; step = 0.1)
vs = predict(model, us)

scatter(xs, ys)
plot!(us, vs, legend=false)
```

✓ 0.0s