Reducing the running time for `gwr.multiscale` in **GWmodel**

Hjalmar Höglund Claudia von Brömssen

August 19, 2025

Abstract

GWmodel is an R package for geographically weighted regression. It allows researchers to consider environmental data collected at different geographic locations and study connections between them. As the number of observations grow, the time to fit these models increases to the point where it is no longer practical for the researcher to run it on their own laptop, instead having to offload the computation to a supercomputer. By reducing memory copies, vectorising the calculations and using memoization the running time is reduced from days to minutes, increasing the size of data sets that can feasibly be handled on a researcher's laptop.

Acknowledgements

The computations on supercomputer infrastructure were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS). This study was funded by the Swedish Research Council for Environment, Agricultural Sciences and Spatial Planning (Formas) (Grant No. 2022-00942).

Report / Department of Energy and Technology, SLU
Report nr: 130
ISSN: 1654-9406

Department of Energy and Technology, SLU
Box 7032
750 07 Uppsala

https://www.slu.se/en/departments/energy-technology/

Uppsala, August 2025

1 Introduction

Identifying relevant drivers for environmental change is crucial to understand processes and decide adequate countermeasures. In order to be able to connect prevailing trends to one or several potential drivers there is a need of datasets that cover a large range of different scenarios, i.e. combinations of different explanatory variables. However, often when the goal of a monitoring program is the quantification of trends a smaller number of sites is selected in order to be able to describe temporal changes well. There are several monitoring programs in Sweden that have good spatial representation. For example, the Swedish Lake survey includes 4800 randomly selected lakes within the national monitoring program and additional 1000 lakes added by regional programs. The Swedish Forest Soil Inventory includes 20000 plots across Sweden and covers all land types except urban areas and cultivated land.

While these datasets have good spatial representation and both chemistry and characteristics of the catchment or surroundings are readily available, the estimation of temporal trends is not straightforward as both these surveys are revisit surveys with a revisit time of 6 years (lakes) and 10 years (forest soils), which means they do not produce typical time series data and the quantification of temporal trends is difficult to conduct for single sites, due to the sparse data. To allow a statistical analysis for such spatially dense but temporally sparse designs geographically weighted regression models (Brunsdon et al., 1998; Gollini et al., 2015) were suggested to provide temporal trends in geographical windows (von Brömssen et al., 2023). The suggested procedure can be used to define where in space temporal trends arise and how strong they are on average within the window.

In a next step of the analysis, it is interesting to determine if trends are connected to local or regional drivers. A logical expansion would be to include one or several potential drivers into the geographically weighted regression models, to study if the trends seen in the response variable change in location or magnitude. Such procedure is sometimes called normalization (often in the context of removing weather driven variation from observed time series data, Stålnacke and Grimvall (2001)). Allowing for additional explanatory variables, beyond time, in GWRs can be done in several ways. Either the coefficient for the explanatory variable is assumed to have the same spatial scale, i.e. it works on the same local/regional scale as time, which allows the use of the basic GWR model (Brunsdon et al., 1998), or the coefficient is assumed to be spatially constant, i.e. only one global estimation of this coefficient is made; mixed GWR (Brunsdon et al., 1998; Mei et al., 2004). A more attractive solution is multiscale GWR, that allow different bandwidth for different variables (A. Stewart Fotheringham and Kang, 2017; Lu et al., 2018; Yang, 2014). Such bandwidth determination could be important if drivers work on different spatial scales and would allow more relevant interpretation of the drivers' effects and severity.

Multiscale GWR is implemented in the package **GWmodel** (Gollini et al., 2015; Binbin Lu and Brunsdon, 2014) in R (R Core Team, 2024), but proved to be slow when used with larger datasets such as the Swedish lake and forest

soil surveys. In this article, we describe and test computational changes to the implementation of multiscale GWR with the goal to decrease computation time to make these models runnable on standard PCs even for moderately large datasets. As a consequence, we see that multiscale GWR might become a standard tool for quantifying environmental change both for revisit designs and for other types of data, where a regional approach is advantageous.

1.1 Programming languages and GWmodel

The programming language C++ is generally considered to be faster than the programming language R, since C++ is compiled whereas R is an interpreted language. The R package **Rcpp** (Eddelbuettel et al., 2024a; Eddelbuettel and François, 2011; Eddelbuettel, 2013; Eddelbuettel and Balamuta, 2018) enables programmers to write routines for R in C++. This is how some parts of the **GWmodel** package are written.

In C++, there are multiple options to achieve parallelism, a process where a problem can be distributed across multiple processing cores working to solve a problem together. The options used in **GWmodel** are OpenMP and CUDA. OpenMP is generally used to write parallelised code for the CPU and CUDA is used to write parallelised code for NVIDIA GPUs.

Two aspects of the `gwr.multiscale` routine are chosen to be optimised. The first is reducing the number of times data is copied from the R runtime to the compiled C++ code, as well as between functions in C++. The second is how the linear algebra is implemented.

1.2 Measuring performance

The performance of the `gwr.multiscale` routine depends on what computer the researcher is using. Users of **GWmodel** can run their analysis on a wide range of computers, spanning from their laptops or desktops to supercomputers. Since our goal is to increase the problem sizes that can be considered on the researcher's computer, we constrain our testing to laptops.

Although the routine `gwr.multiscale` will be the main focus for our work, some subroutines are shared with other routines. `gwr.basic`, a routine for performing geographical weighted regression for a single explanatory variable, and `bw.gwr`, a routine for finding the bandwidth for a variable, also benefit from our optimisations.

2 Method

Performance improvement can be achieved by limiting the number of times data has to be copied. To achieve this, a sufficiently large part of the `gwr.multiscale` routine was implemented in C++, such that only a single call to the C++ code is performed.

To further reduce the number of times data is copied, parameters to functions were sent by reference as opposed to sending them by value in the C++ code. In the case of a matrix being a parameter to a C++ function `foo`, this would change the function definition from

```
double foo(matrix mat);
```

to

```
double foo(const matrix &mat);
```

Another way to speed up computation is to *vectorise* the computation. By changing kernel functions to operate on vectors instead of single matrix elements, the number of function calls is reduced. Additionally, we get some speed up by being able to leverage *SIMD*, single instruction, multiple data. This means that calls to kernel functions went from looking like this

```
for (int c = 0; c < numcolumns; c++) {
  for (int r = 0; r < numrows; r++) {
    result(r,c) = kernel_func(mat(r,c));
  }
}
```

to instead look like this

```
for (int c = 0; c < numcolumns; c++) {
  result.column(c) = kernel_func_vec(mat.column(c));
}
```

To reduce the time consumed by linear algebra operations, the linear algebra library used was swapped from **RcppArmadillo** (Eddelbuettel et al., 2024b; Eddelbuettel and Sanderson, 2014) to **RcppEigen** (Bates and Eddelbuettel, 2013). **RcppEigen** has support for running on multicore systems and offers some operations that might speed up the calculations. In addition to this, linear solvers were used to solve linear systems, as opposed to computing matrix inverses. Finally, some memoization was performed to avoid repeating computation.

To test how much faster the code had become, the `gwr.multiscale` routine was run on a collection of datasets varying in size. The original and new implementations were run, and running time was measured using the **tictoc** package (Izrailev, 2024). The time needed to read the data from disk and prepare it for analysis was not included, only the time to fit the model was measured. Three warm-up runs were done, followed by ten real runs. The mean and the standard deviation was recorded.

The tests were repeated on different consumer grade laptops, varying the number of cores on the machine. The machines had different operating systems, Microsoft Windows and Apple macOS.

The datasets used to perform the analysis were taken from the Swedish Lake survey and the Swedish Forest Soil Inventory. To vary the size of the dataset, different cutoffs for the observation x-coordinate were chosen. The number of explanatory variables varied from two to four.

	# observations						
Version	100	200	300	401	500	601	702
New	$1.70 \cdot 10^{-1}$	$3.30 \cdot 10^{-1}$	$5.10 \cdot 10^{-1}$	$8.40 \cdot 10^{-1}$	$1.60 \cdot 10^{0}$	$2.40 \cdot 10^{0}$	$4.30 \cdot 10^{0}$
Original	$4.50 \cdot 10^{-1}$	$2.10 \cdot 10^{0}$	$6.40 \cdot 10^{0}$	$1.60 \cdot 10^{1}$	$3.10 \cdot 10^{1}$	$5.60 \cdot 10^{1}$	$9.20 \cdot 10^{1}$

Table 1: Execution time in seconds on 6 threads. Limited to 10 iterations on laptops.

	# observations						
# threads	100	200	300	401	500	601	702
1	2.6	3.9	5.5	8.5	9.6	11.0	10.1
4	4.7	13.5	19.2	28.6	31.2	35.7	32.3
6	6.5	13.3	25.6	35.7	40.0	47.6	45.5

Table 2: Speedup factor of new implementation compared to original implementation for different problem sizes and number of threads. Limited to 10 iterations run on laptop.

To ensure that the new code produces the same output as the original code, comparisons between the produced outputs from both versions were performed, with the maximum relative difference recorded.

3 Results

The running times of the original and the new implementation for different problem sizes are presented in table 1. The speedup factors of the new implementation compared to the original implementation for different problem sizes and configurations are presented in tables 2 and 3. The running time for a larger problem size is presented in table 4.

The maximum relative difference ranged from 1% to 10 ppm when considering all but one kernel function. When the kernel function *boxcar* was chosen the largest relative error was 10%.

The running time for the routines `gwr.basic` and `bw.gwr` are presented in tables 5 and 6.

4 Discussion

A substantial speedup was achieved following our modification of the code. The implications of this speedup are that larger datasets become viable for analysis and that a larger variety of computer hardware can be used to run the code.

The main reason for the speedup is believed to be reduced copying of memory, and switching to a library for linear algebra that supports parallel matrix operations. Along with these changes some other optimisations were performed,

	# observations		
# variables	301	501	701
2	20.8	17.9	38.5
3	21.3	34.5	45.5
4	21.7	40.0	52.6

Table 3: Speedup factor of new implementation compared to original implementation for different number of explanatory variables on 6 threads. Limited to 10 iterations run on laptops.

	Model 1		Model 2	
Version	μ	σ	μ	σ
New	$2.04 \cdot 10^2$	$1.80 \cdot 10^1$	$2.01 \cdot 10^2$	$2.57 \cdot 10^1$
Original	$8.92 \cdot 10^4$	-	$8.89 \cdot 10^4$	-

Table 4: Average execution time of new implementation on dataset with 4939 observations on 4 threads limited to 10 iterations run on laptops. Both models use two explanatory variables. Average of 5 runs for new implementation, old implementation only did a single run.

	Adaptive		Fixed	
Version	μ	σ	μ	σ
New	$4.41 \cdot 10^0$	$1.55 \cdot 10^{-1}$	$2.05 \cdot 10^0$	$1.53 \cdot 10^{-1}$
Original	$9.12 \cdot 10^0$	$3.16 \cdot 10^{-1}$	$3.39 \cdot 10^0$	$1.93 \cdot 10^{-1}$

Table 5: Average execution time of `gwr.basic` on dataset with 11730 observations on 4 threads limited to 10 iterations run on laptops. Average of 10 runs.

	CV & Adaptive		CV & Fixed		AIC & Adaptive		AIC & Fixed	
Version	μ	σ	μ	σ	μ	σ	μ	σ
New	$7.59 \cdot 10^0$	$6.62 \cdot 10^{-1}$	$1.25 \cdot 10^1$	$1.77 \cdot 10^0$	$1.82 \cdot 10^1$	$1.86 \cdot 10^0$	$2.13 \cdot 10^1$	$2.49 \cdot 10^0$
Original	$3.81 \cdot 10^1$	$6.42 \cdot 10^{-1}$	$4.44 \cdot 10^1$	$9.84 \cdot 10^{-1}$	$4.80 \cdot 10^1$	$3.71 \cdot 10^{-1}$	$3.34 \cdot 10^1$	$7.23 \cdot 10^{-1}$

Table 6: Average execution time of `bw.gwr` on dataset with 11730 observations on 4 threads limited to 10 iterations run on laptops. Average of 10 runs.

such as memoizing calculations to avoid repetition. In addition to the quicker execution time, numerical accuracy has increased by switching to linear solvers rather than computing matrix inverses.

Some values differed greatly when using the *boxcar* kernel function. This is believed to be caused by the increased stability obtained by using linear solvers rather than computing the matrix inverse. Small numeric differences are amplified by the use of a kernel function which cuts off any points outside the bandwidth, thereby giving a large difference in the final result.

4.1 Future work

GWmodel currently supports CUDA, a parallelisation framework that can be used on NVIDIA GPUs. An alternative to optimising the CUDA implementation would be to support some other framework for offloading computation to the GPU, like OpenCL or HIP. Adding this support would remove the requirement to use GPUs from NVIDIA and enable anyone with a GPU from some other manufacturer to offload their CPU.

There is however a negative side effect with extending support to include other frameworks. With the addition of OpenCL or HIP, the number of parallelisation frameworks is increased, and so will the burden of maintaining the package. Extending the OpenMP code to instead support computation on the GPU might be preferable.

Another avenue to explore is improving the fitting routine by trying other techniques to find the optimal bandwidth for each variable. This is currently done using a golden section search. We have not yet ruled out that there might be a more efficient approach to finding the optimal bandwidth, such as using the *Newton-Raphson method*.

When working on a laptop or other personal computer, the amount of available memory is what limits what problem sizes can be considered. We have explored performing the computation using single precision (32-bit) rather than double precision (64-bit). This makes the data take up less space in memory, but reduces the precision. For some test, the reduced precision leds to routines producing different outputs, indicating that this might not always be viable. Exploring ways to determine when to switch to single precision could offer a reduction in memory footprint.

5 Conclusion

The rewrite has increased the size of datasets that can be analysed by the routine and made it better suited for parallel systems. The shorter execution time allows researchers to consider geographically weighted regression in cases where waiting for model fitting would have previously been prohibitive.

References

W. Y. A. Stewart Fotheringham and W. Kang. Multiscale geographically weighted regression (mgwr). *Annals of the American Association of Geographers*, 107(6):1247–1265, 2017. doi: 10.1080/24694452.2017.1352480. URL https://doi.org/10.1080/24694452.2017.1352480.

D. Bates and D. Eddelbuettel. Fast and elegant numerical linear algebra using the RcppEigen package. *Journal of Statistical Software*, 52(5):1–24, 2013. doi: 10.18637/jss.v052.i05.

M. C. Binbin Lu, Paul Harris and C. Brunsdon. The gwmodel r package: further topics for exploring spatial heterogeneity using geographically weighted models. *Geo-spatial Information Science*, 17(2):85–101, 2014. doi: 10.1080/10095020.2014.917453. URL https://doi.org/10.1080/10095020.2014.917453.

C. Brunsdon, S. Fotheringham, and M. Charlton. Geographically weighted regression. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 47(3):431–443, 1998. doi: https://doi.org/10.1111/1467-9884.00145. URL https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/1467-9884.00145.

D. Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. Springer, New York, 2013. doi: 10.1007/978-1-4614-6868-4. ISBN 978-1-4614-6867-7.

D. Eddelbuettel and J. J. Balamuta. Extending R with C++: A Brief Introduction to Rcpp. *The American Statistician*, 72(1):28–36, 2018. doi: 10.1080/00031305.2017.1375990.

D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. doi: 10.18637/jss.v040.i08.

D. Eddelbuettel and C. Sanderson. Rcpparmadillo: Accelerating r with high-performance c++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063, March 2014. doi: 10.1016/j.csda.2013.02.005.

D. Eddelbuettel, R. Francois, J. Allaire, K. Ushey, Q. Kou, N. Russell, I. Ucar, D. Bates, and J. Chambers. *Rcpp: Seamless R and C++ Integration*, 2024a. URL https://CRAN.R-project.org/package=Rcpp. R package version 1.0.12.

D. Eddelbuettel, R. Francois, D. Bates, B. Ni, and C. Sanderson. *RcppArmadillo: 'Rcpp' Integration for the 'Armadillo' Templated Linear Algebra Library*, 2024b. URL https://CRAN.R-project.org/package=RcppArmadillo. R package version 0.12.8.4.0.

I. Gollini, B. Lu, M. Charlton, C. Brunsdon, and P. Harris. Gwmodel: An r package for exploring spatial heterogeneity using geographically weighted

models. *Journal of Statistical Software*, 63(17):1–50, 2015. doi: 10.18637/jss.v063.i17. URL https://www.jstatsoft.org/index.php/jss/article/view/v063i17.

S. Izrailev. *tictoc: Functions for Timing R Scripts, as Well as Implementations of "Stack" and "StackList" Structures*, 2024. URL https://CRAN.R-project.org/package=tictoc. R package version 1.2.1.

B. Lu, W. Yang, Y. Ge, and P. Harris. Improvements to the calibration of a geographically weighted regression with parameter-specific distance metrics and bandwidths. *Computers, Environment and Urban Systems*, 71:41–57, 2018. ISSN 0198-9715. doi: https://doi.org/10.1016/j.compenvurbsys.2018.03.012. URL https://www.sciencedirect.com/science/article/pii/S0198971517303447.

C.-L. Mei, S.-Y. He, and K.-T. Fang. A note on the mixed geographically weighted regression model*. *Journal of Regional Science*, 44(1):143–157, 2004. doi: https://doi.org/10.1111/j.1085-9489.2004.00331.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1085-9489.2004.00331.x.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2024. URL https://www.R-project.org/.

P. Stålnacke and A. Grimvall. Semiparametric approaches to flow normalization and source apportionment of substance transport in rivers. *Environmetrics*, 12(3):233–250, 2001. doi: https://doi.org/10.1002/env.459. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/env.459.

C. von Brömssen, J. Fölster, and K. Eklöf. Temporal trend evaluation in monitoring programs with high spatial resolution and low temporal resolution using geographically weighted regression models. *Environmental Monitoring and Assessment*, 195(5):547, 2023. doi: 10.1007/s10661-023-11172-2. URL https://doi.org/10.1007/s10661-023-11172-2.

W. Yang. *An extension of geographically weighted regression with flexible bandwidths*. Phd thesis, University of St Andrews, 2014. Available at https://hdl.handle.net/10023/7052.

A Computer specifications

Below is the specifications for the computers used to measure running time.

Processor	Cores	Mem. size	Operating system
Apple M1 Pro	8 (6 perf.)	16 GB	macOS 15.5
Intel Core i7	8	16 GB	Windows 10
Intel Core Ultra 7 155H	16 (6 perf.)	32 GB	Windows 11

Table 7: Computer specifications for laptops.

B OpenMP on macOS computers

As outlined by the *OpenMP on macOS with Xcode tools* page on the R-project, enabling OpenMP for packages in R is not as straight forward as it is for Windows or for Linux computers. We followed the instructions on the page when installing both our new code and the original code. During the testing, we also validated that both versions of the code used multiple cores.