Non Parametric Feature Engineering for Machine Learning

by

Sumit Mahaveer Sethi

A project submitted in partial fulfillment

of the requirements for the degree of

Master in Financial Engineering

Department of Finance and Risk Engineering

New York University

August, 2020

Instructed by

Dr. Jerzy Pawlowski

ABSTRACT

This capstone project features the implementation of a library of C++ functions for calculating non-parametric estimators of time series data. The estimators can be used for feature engineering for machine learning applications. The library also implements fast rolling functions over time series data.

CONTENTS

LIST OF FIGURES

1 | INTRODUCTION

The standard statistical estimators of the moments (mean, variance, skewness) are often used as features in machine learning models, but they are not always well suited as features. First, because financial time series data is often far from normally distributed, which violates the assumptions of many models, leading to the underestimation of the standard errors of predictions. Secondly, standard estimators are not the most efficient for skewed distributions in the presence of noise. On the other hand, nonparametric estimators are more robust to noise and can offer a better bias-variance tradeoff. But nonparametric estimators often require calculating the sorts, ranks, and the quantiles of data, which are time consuming. It's therefore better to implement them using fast C++ functions, rather than using Python or R. In addition, the nonparametric estimators need to be applied over a rolling time window. This can be achieved by applying the nonparametric estimators in a C++ loop over the time series data.

The capstone project implements a variety of nonparametric estimators, including estimators of location (median, Hodges-Lehmann), of dispersion (Median Absolute Deviation), of skewness (medcouple), and of dependency-covariance (Theil-Sen). It also implements the nonparametric statistics of the Wilcoxon Signed Rank test, the Mann-Whitney-Wilcoxon Rank Sum test, and the Kruskal-Wallis test. Standardized PCA is also implemented. Many of these statistics are already implemented, but they are not easily applied over a rolling time window in an efficient way.

The emphasis in this project is on achieving very fast computation speeds. Parallel processing is employed on multi-core CPUs, to further accelerate the calculations.

2 | Non Parametric Estimators

2.1 Location Estimators

The fundamental task in many statistical analysis is to estimate a location parameter for the distribution; i.e. to find typical or central value that best describes the data. The standard estimator of the location is Mean. But in non-normal distributions the mean can be skewed due to outliers and it may not be the accurate representation of the location of distribution. The non- parametric estimators like Hodges-Lehmann Estimator and Median will do better job at describing data in case of non-normal distributions.

2.1.1 Hodges-Lehmann Estimator

The Hodges–Lehmann estimator is a robust and non parametric estimator of a population's location parameter. Its computation can be described quickly. For a data set with n measurements, the set of all possible one- or two-element subsets of it has n(n + 1)/2 elements. For each such subset, the mean is computed; finally, the median of these n(n + 1)/2 averages is defined to be the Hodges–Lehmann estimator of location.

2.1.1.1 Implementation

Hodges-Lehmann estimator of location for a vector or a single-column time series is implemented in C++ using Rcpp, RcppArmadillo and RcppParallel packages.

$$h = NPE :: hle(vector)$$

where h is the Hodges-Lehmann estimator value for vector or single column time series.

2.1.2 Median

A median of a population is any value such that at most half of the population is less than the proposed median and at most half is greater than the proposed median.

2.1.2.1 Implementation

Median is also an estimator of location for a vector or a single-column time series is implemented in C++ using Rcpp and RcppArmadillo packages. There is also rolling window implementation of Median which calculates rolling medians in given window over a time series or vector.

$$m = NPE :: med\_ian(vector)$$

where m is the median value for vector or single column time series.

$$m = NPE :: rolling\_median(vector, window)$$

where m is the vector of rolling median over given window for vector or single column time series.

2.2 Dispersion Estimators

Dispersion (also called variability, scatter, or spread) is the extent to which a distribution is stretched or squeezed. Standard Estimators for the dispersion are Standard Deviation or Variance. But these estimators does not represent the dispersion of distribution accurately in case distribution is Non- normal. In such cases, we can use non-parametric dispersion estimators like Median Absolute Deviation.

2.2.1 Median Absolute Deviation

In statistics, the median absolute deviation (MAD) is a robust measure of the variability of a univariate sample of quantitative data. It can also refer to the population parameter that is estimated by the MAD calculated from a sample.

For a univariate data set $X_1, X_2, ..., X_n$, the MAD is defined as the median of the absolute deviations from the data's median $\tilde{X}$ = median(X)

$$MAD = median(|X_i - \tilde{X}|)$$

that is, starting with the residuals (deviations) from the data's median, the MAD is the median of their absolute values.

2.2.1.1 Implementation

Median Absolute deviation(MAD) of a vector or a single-column time series is implemented in C++ using Rcpp and RcppArmadillo packages. There is also rolling window implementation of MAD over a time series or vector.

$$m = NPE :: medianAbsoluteDeviation(vector)$$

4

where m is the medianAbsoluteDeviation value for vector or single column time series.

$$m = NPE :: rolling\_mad(vector, window)$$

where m is the vector of rolling MAD over given window for vector or single column time series.

2.3 ESTIMATOR OF SKEWNESS

Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. The Skewness value can be positive, zero, negative, or undefined. The non parametric variant of the skewness is Medcouple.

2.3.1 MEDCOUPLE

The medcouple is a robust statistic that measures the skewness of a univariate distribution. It is defined as a scaled median difference of the left and right half of a distribution.

2.3.1.1 IMPLEMENTATION

Medcouple estimator of skewness for a vector or a single-column time series is implemented in C++ using Rcpp, RcppArmadillo packages.

$$mc = NPE :: med\_couple(vector)$$

where mc is the medcouple of a vector or single colunm time series.

2.4 Theil-Sen Estimator for dependency covariance

The Theil–Sen estimator is a method for robustly fitting a line to sample points in the plane (simple linear regression) by choosing the median of the slopes of all lines through pairs of points. It has also been called Sen's slope estimator, slope selection, the single median method, the Kendall robust line-fit method, and the Kendall–Theil robust line.

2.4.0.1 Implementation

Theil-Sen Estimator for a vector or a single-column time series is implemented in C++ using Rcpp, RcppArmadillo packages.

$$ts = NPE :: theilSenEstimator(vec\_x, vec\_y)$$

where ts is the intercept and slope calculated by Theil-Sen Estimator for a vector or single colunm time series.

3 | Non Parametric Statistics

3.1 Wilcoxon Signed Rank test

The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test used to compare two related samples, matched samples, or repeated measurements on a single sample to assess whether their population mean ranks differ (i.e. it is a paired difference test). It can be used as an alternative to the paired Student's t-test (also known as "t-test for matched pairs" or "t-test for dependent samples") when the distribution of the difference between two samples' means cannot be assumed to be normally distributed. A Wilcoxon signed-rank test is a nonparametric test that can be used to determine whether two dependent samples were selected from populations having the same distribution.

Implementation

Wilcoxan Signed Rank Test implementation is for one sample test on a vector or single column time series using Rcpp, RcppArmadillo and Boost packages.

$$wilcoxanSignedRankTest(x, mu = 0, alternative = "two.sided", exact = FALSE, correct = TRUE)$$

where, x is a vector or a single-column time series.

mu is a double specifying an optional parameter used to form null hypothesis with Default value

set to zero.

alternative is a character string specifying the alternative hypothesis. It must be one of : "two.sided"
two tailed test. "greater" greater(right) tailed test. "less" smaller(left) tailed test. (The default for
alternative is two.sided test.)

exact is a boolean indicating whether an exact p-value should be computed.

correct is a boolean indicating whether to apply continuity correction in normal approximation
for the p-value.

This function returns the p value of the test.

3.2   Wilcoxon-Mann-Whitney Signed Rank test

the Mann–Whitney–Wilcoxon (MWW) (also called the Mann–Whitney U test, Wilcoxon rank-
sum test, or Wilcoxon–Mann–Whitney test) is a nonparametric test of the null hypothesis that the
probability that a randomly selected value from one population is less than a randomly selected
value from a second population is equal to the probability of being greater.

This test can be used to investigate whether two independent samples were selected from
populations having the same distribution. The Mann-Whitney U test is often used when the
assumptions of the independent samples t-test are violated. This test is simlilar to the Wilcoxon
signed-rank test used on single sample or dependent samples.

Implementation

Wilcoxan-Mann-Whitney Signed Rank Test implementation is for two sample test on a vector or
single column time series using Rcpp, RcppArmadillo and Boost packages.

$wilcoxanMannWhitneyTest(x, y, mu = 0, alternative = "two.sided", exact = FALSE, correct = TRUE)$

where, x and y are two independent vectors or a single-column time series.

mu is a double specifying an optional parameter used to form null hypothesis with Default value set to zero.

alternative is a character string specifying the alternative hypothesis. It must be one of : "two.sided" two tailed test. "greater" greater(right) tailed test. "less" smaller(left) tailed test. (The default for alternative is two.sided test.)

exact is a boolean indicating whether an exact p-value should be computed.

correct is a boolean indicating whether to apply continuity correction in normal approximation for the p-value.

This function returns the p value of the test.

3.3   KRUSKAL-WALLICE TEST

The Kruskal-Wallis test is a nonparametric (distribution free) test, and is used when the assumptions of one-way ANOVA are not met. Both the Kruskal-Wallis test and one-way ANOVA assess for significant differences on a continuous dependent variable by a categorical independent variable (with two or more groups). In the ANOVA, we assume that the dependent variable is normally distributed and there is approximately equal variance on the scores across groups. However, when using the Kruskal-Wallis Test, we do not have to make any of these assumptions. Therefore, the Kruskal-Wallis test can be used for both continuous and ordinal-level dependent variables. However, like most non-parametric tests, the Kruskal-Wallis Test is not as powerful as the ANOVA.

Null hypothesis: Null hypothesis assumes that the samples (groups) are from identical populations.

Alternative hypothesis: Alternative hypothesis assumes that at least one of the samples (groups) comes from a different population than the others.

Implementation

Kruskal Wallice Test implementation is for list of the vectors or single column time series using Rcpp, RcppArmadillo and Boost packages.

$$kruskalWalliceTest(x)$$

where, x is a list of the numeric data vectors.

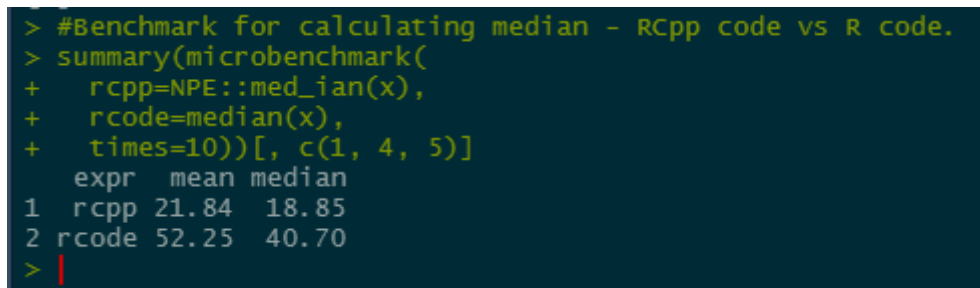This function returns the p value of the test.

4 | RESULTS

4.1 BENCHMARKING OF NPE FUNCTIONS

The C++ library is part of a R package **NPE**, allowing users to easily call the C++ functions from R. The R environment will serve as the user interface for the C++ library.

One of the objectives of this library it to provide highly efficient and faster implementations of non-parametric functions. We benchmark these functions against their counterparts in R using package microbenchmark.

4.1.1 MEDIAN

We benchmark **NPE::med_ian** function against R function **median** on array of 100 random numbers.

```
> #Benchmark for calculating median - RCpp code vs R code.
> summary(microbenchmark(
+   rcpp=NPE::med_ian(x),
+   rcode=median(x),
+   times=10))[, c(1, 4, 5)]
   expr  mean median
1  rcpp 21.84  18.85
2 rcode 52.25  40.70
>
```

Figure 4.1: Median : NPE Vs R implementation

As we can see in Figure 4.1, The NPE function is more than twice faster than R implementation.

4.1.2 Hodges Lehman Estimator

We benchmark **NPE::hle** function against R function **wilcox.test** on array of 100 random numbers.

```
> #Hodges-Lehmann Estimator
> x <- runif(100)  #above 50 wilcox.test function will approximate the results.
> wilcox.test(x, conf.int = TRUE)$estimate
(pseudo)median
     0.4341895
> NPE::hle(x)
[1] 0.4342436
> all.equal((NPE::hle(x))[1], wilcox.test(x, conf.int = TRUE)$estimate)
[1] "names for current but not for target"  "Mean relative difference: 0.000124684"
> summary(microbenchmark(
+    RCpp=NPE::hle(x),
+    R=wilcox.test(x, conf.int = TRUE)$estimate,
+    times=10))[, c(1, 4, 5)]
  expr      mean    median
1 RCpp  1.83436   1.85890
2    R 13.51021  13.44165
>
```

Figure 4.2: Hodges-Lehmann Estimator : NPE Vs R implementation

As we can see in Figure 4.2, The NPE function is seven to eight times faster than R implementation.

4.1.3 Median Absolute Deviation

We benchmark **NPE::medianAbsoluteDeviation** function against R function **mad** on array of 10 random numbers.

```
> #Benchmark for calculating Median absolute deviation - RCpp code vs R code.
> summary(microbenchmark(
+    RCpp = NPE::medianAbsoluteDeviation(x),
+    R=mad(x, constant = 1),
+    times=10))[, c(1, 4, 5)]
  expr   mean median
1 RCpp 22.31  17.35
2    R 92.41  80.70
>
```

Figure 4.3: Median Absolute Deviation : NPE Vs R implementation

As we can see in Figure 4.3, The NPE function is four times faster than R implementation.

4.1.4 Medcouple

We benchmark **NPE::med_couple** function against **mc** function in package **robustbase** which

is implemented in C.

```
> # Medcouple
> library(robustbase)
> #Rcpp::sourceCpp(file = "E:\\Summer term\\project\\test.cpp")
> x1 <- c(1, 2, 7, 9, 10)
> NPE::med_couple(x1)
[1] -0.3333333
> robustbase::mc(x1)
[1] -0.3333333
> all.equal(NPE::med_couple(x1), robustbase::mc(x1))
[1] TRUE
> x <- c(1:5,7,10,15,25)
> summary(microbenchmark(
+    rcpp=NPE::med_couple(x),
+    robustbase= robustbase::mc(x),
+    times=10))[, c(1, 4, 5)]
       expr  mean median
1      rcpp 20.42  16.45
2 robustbase 47.87  34.35
```

Figure 4.4: Medcouple : NPE Vs R implementation

As we can see in Figure 4.4, The NPE function is more than twice times faster than robustbase

implementation.

4.1.5 Theil-Sen Estimator

We benchmark **NPE::theilSenEstimator** function against R function **tsreg** from package **WRS**.

```
> #Theil-Sen Estimator
> x <- runif(10)
> y <- runif(10)
> library("WRS")
> tsreg(x, y)$coef    #there is very small difference in intercept because WRS package adjusts it for residuals and I don't.
 Intercept
 0.5995084 -0.3956290
> NPE::theilSenEstimator(x, y)
[1]  0.6119171 -0.3956290
> summary(microbenchmark(
+    RCpp=NPE::theilSenEstimator(x, y),
+    R=tsreg(x, y)$coef,
+    times=10))[, c(1, 4, 5)]
   expr    mean median
1 RCpp  36.51  33.45
2    R 579.75 571.95
>
```

Figure 4.5: Theil-Sen Estimator : NPE Vs R implementation

As we can see in Figure 4.5, The NPE function is more than sixteen times faster than R implementation.

4.1.6 PCA

We benchmark **NPE::calc_pca** function against R function **prcomp**.

Figure 4.6: PCA : NPE Vs R implementation

As we can see in Figure 4.6, The NPE function is more than six times faster than R implementation.

4.1.7 WILCOXON RANKED SUM TEST

We benchmark **NPE::wilcoxonRankedSumTest** function against R function **wilcox.test**.

Figure 4.7: Wilcoxon Ranked Sum Test : NPE Vs R implementation

As we can see in Figure 4.7, The NPE function is almost four times faster than R implementation.

4.1.8 Wilcoxon-Mann-Whitney Test

We benchmark **NPE::wilcoxonMannWhitneyTest** function against R function **wilcox.test**.

```
> #wilcoxan-Mann_whitney rank sum test
> x <- c(0.80, 0.83, 1.89, 1.04, 1.45, 1.38, 1.91, 1.64, 0.73, 1.46)
> y <- c(1.15, 0.88, 0.90, 0.74, 1.21)
> all.equal(wilcox.test(x, y, alternative = "two.sided")$p.value, NPE::wilcoxanMannWhitneyTest(x, y, alternative = "two.sided"))
[1] TRUE
> wilcox.test(x, y, alternative = "two.sided")$p.value
[1] 0.2544123
> NPE::wilcoxanMannWhitneyTest(x, y, alternative = "two.sided")
[1] 0.2544123
> summary(microbenchmark(
+   RCpp=NPE::wilcoxanMannWhitneyTest(x, y, alternative = "two.sided"),
+   R=wilcox.test(x, y, alternative = "greater")$p.value,
+   times=10))[, c(1, 4, 5)]
  expr   mean median
1 RCpp  54.32  46.35
2    R 125.26 109.15
>
```

Figure 4.8: Wilcoxon-Mann-Whitney Ranked Sum Test : NPE Vs R implementation

As we can see in Figure 4.8, The NPE function is more than twice faster than R implementation.

4.1.9 Kruskal-Wallice test

We benchmark **NPE::kruskalWalliceTest** function against R function **kruskal.test**.

```
> #Kruskal-Wallice test.
> x <- c(2.9, 3.0, 2.5, 2.6, 3.2)
> y <- c(3.8, 2.7, 4.0, 2.4)
> z <- c(2.8, 3.4, 3.7, 2.2, 2.0)
> kruskal.test(list(x, y, z))$p.value
[1] 0.6799648
> NPE::kruskalWalliceTest(list(x, y, z))
[1] 0.6760903
> summary(microbenchmark(
+   RCpp=NPE::kruskalWalliceTest(list(x, y, z)),
+   R=kruskal.test(list(x, y, z))$p.value,
+   times=10))[, c(1, 4, 5)]
  expr   mean median
1 RCpp  32.30  29.85
2    R 334.74 320.05
```

Figure 4.9: Kruskal-Wallice Test : NPE Vs R implementation

As we can see in Figure 4.9, The NPE function is almost ten times faster than R implementation.

4.2 Non Parametric Estimators Vs Standard Estimators in Empirical Time Series Data

Second objective of this project is to compare non parametric estimators with standard estimators for time series data. For this comparison we are using **Empirical time series of Financial Select Sector SPDR Fund ("XLF")** from 04-17-2000 to 04-09-2020.

4.2.1 Location Estimators

We applied Standard Estimators of the Location (Mean) and Non Parametric estimators like Median and Hodges-Lehmann Estimator to the time series data. Then calculated their respective standard errors using Bootstrap Simulations.

```
> # Mean and standard errors of location estimators from bootstrap
> boot_data <- sapply(boot_sample, function(sampl_e) {
+   c(median_est=NPE::med_ian(sampl_e),
+     hl_est = NPE::hle(sampl_e),
+     mean_est=mean(sampl_e))
+ })  # end sapply
> apply(boot_data, MARGIN=1, function(x)
+   c(mean=mean(x), std_error=sd(x)))
            median_est       hl_est      mean_est
mean      0.0018147404 0.0019729280 0.0021959703
std_error 0.0008543518 0.0009147974 0.0009080589
> |
```

Figure 4.10: Location Estimators : Non Parametric Vs Standard Estimators

As we can see in Figure 4.10, the median Estimator have the lowest Standard Error, slightly lower than the Hodges-Lehmann. i.e median is doing better job at describing the location of the data it's standard counterpart (Mean).

4.2.2 Dispersion Estimators

Similarly, We applied Standard Estimators of the Dispersion (Standard Deviation) and Non Parametric estimators like Median Absolute Deviation to the time series data. Then calculated their

respective standard errors using Bootstrap Simulations.

```
> # Mean and standard error of MAD estimator from bootstrap
> boot_data <- sapply(boot_sample, function(sampl_e) {
+   c(mad_est=NPE::calc_mad(sampl_e), std_dev=sd(sampl_e))
+ })   # end sapply
> apply(boot_data, MARGIN=1, function(x)
+   c(mean=mean(x), std_error=sd(x)))
              mad_est        std_dev
mean       0.0050443651 0.0123816712
std_error  0.0001042076 0.0003309916
```

Figure 4.11: Dispersion Estimators : Non Parametric Vs Standard Estimators

As we can see in Figure 4.11, the Median Absolute Deviation have the lowest Standard Error.
i.e it is doing better job at describing the dispersion of the data than it's standard counterpart
(Standard Deviation).

4.2.3   SKEWNESS ESTIMATORS

Similarly, We applied Parametric Estimators of the Skewness (Pearson Skewness) and Non Para-
metric estimators like Medcouple to the time series data. Then calculated their respective stan-
dard errors using Bootstrap Simulations.

```
> # Mean and standard error of medcouple estimator from bootstrap
> boot_data <- sapply(boot_sample, function(sampl_e) {
+   c(med_couple=NPE::med_couple(sampl_e),
+     pearson_skew=NPE::calc_skew(sampl_e, typ_e="Pearson"))
+ })   # end sapply
> apply(boot_data, MARGIN=1, function(x)
+   c(mean=mean(x), std_error=sd(x)))
            med_couple pearson_skew
mean       -0.03235226   -0.1289659
std_error   0.01876925    0.3823784
```

Figure 4.12: Skewness Estimators : Non Parametric Vs Standard Estimators

As we can see in Figure 4.12, the Medcouple have the lowest Standard Error. i.e it is doing better
job at describing the skew of the data than it's standard counterpart (Skewness).

We also compared Pearson Skewness, Quantile Skewness and Nonparametric Skewness.
As we can see in Figure 4.13, the Non-parametric have the lowest Standard Error. i.e it is doing
better job at describing the skew of the data than it's parametric counterparts.

Figure 4.13: Skewness Estimators : Parametric Vs Non Parametric Estimators

4.3 Package NPE vs RcppRoll vs roll

Functionalities like rolling median are offered by different packages. Even though NPE is unique to offer these many nonparamteric functions in one package, we need to benchmark NPE with these packages too.

NPE is implemented in C++ using Rcpp and for rolling functions it make use of the parallel processing through package RcppParallel.

Similarly package roll is also implemented using Rcpp and RcppParallel, while package RcppRoll does not implement parallel processing but is written in Rcpp as well.

The difference in all these packages is how they calculate the median. Package NPE uses the RcppArmadillo::median Function, package roll uses std::sort on an array and then calculate the median, while RcppRoll uses std::partial_sort_copy.

Due to these differences, Each of these package can be faster than the other depending on the look back window as shown below.

As we can see in Figure 4.14, When look back window is larger(100 in this case), the NPE function is lot faster than the package roll and RcppRoll due to parallel parocessing and optimised median calculating method in RcppArmadillo.

But if the look back window is smaller (10 in this case), NPE is lot slower than package roll, due to different apporch of calculating median. NPE is also slightly slower than RcppRoll, since the

```
> re_turns <- na.omit(NPE::etf_env$re_turns[ ,"VTI"])
> summary(microbenchmark(
+    NPE_rolling_median=NPE::rolling_median(re_turns, look_back=100),
+    roll_roll_median=roll::roll_median(re_turns, width=100),
+    RcppRoll_roll_median = RcppRoll::roll_median(re_turns, 100 ),
+    times=10))[, c(1, 4, 5)]  # end microbenchmark summary
                  expr    mean   median
1   NPE_rolling_median  1.77086  1.78785
2     roll_roll_median  3.10714  3.09315
3 RcppRoll_roll_median 12.96709 12.94630
> re_turns <- na.omit(NPE::etf_env$re_turns[ ,"VTI"])
> summary(microbenchmark(
+    NPE_rolling_median=NPE::rolling_median(re_turns, look_back=10),
+    roll_roll_median=roll::roll_median(re_turns, width=10),
+    RcppRoll_roll_median = RcppRoll::roll_median(re_turns, 10 ),
+    times=10))[, c(1, 4, 5)]  # end microbenchmark summary
                  expr    mean   median
1   NPE_rolling_median 1212.14 1136.50
2     roll_roll_median  255.67  241.95
3 RcppRoll_roll_median  957.13  956.30
> summary(microbenchmark(
+    NPE_rolling_median=NPE::rolling_median(re_turns, look_back=30),
+    roll_roll_median=roll::roll_median(re_turns, width=30),
+    RcppRoll_roll_median = RcppRoll::roll_median(re_turns, 30 ),
+    times=10))[, c(1, 4, 5)]  # end microbenchmark summary
                  expr    mean   median
1   NPE_rolling_median 1387.13 1368.30
2     roll_roll_median 1040.18 1038.25
3 RcppRoll_roll_median 3286.59 3333.30
```

Figure 4.14: NPE Vs roll Vs RcppRoll : rolling_median functions

overhead of parallel processing is not compensated due to the small look back values(less data to process on each thread).

For mid sized look back window (30), NPE is almost similar to roll and lot faster than RcppParallel.

5 | Conclusion

The Package **NPE** created under this project provides the faster implementations of the Non-Parametric Estimators as well as statistics than the currently available R functions by making use of C++ and parallel programming. We also applied These nonparametric estimators to empirical time series data. Their standard errors were estimated using bootstrap simulation, and they were compared to those of standard estimators, to demonstrate that nonparametric estimators offer a better bias-variance tradeoff.

A | Appendix

A.1 Installation Guide

To install Package NPE, you need to have packages Rcpp, RcppParallel, RcppArmadillo and BH.

Installation Commands:

install.packages("devtools")

devtools::install_github(repo = "marvic24/Non-Parametric-Estimators")

Loading the package:

library(NPE)