Runtime Benchmarking

Contents

Introduction

In these simulations, we explore how the run time for the `amkat` function from the `AMKAT` package scales with the sample size and feature dimension of the data. We perform benchmarking of the execution time for `amkat` with PhiMr compared to without PhiMr, both to evaluate the impact of PhiMr on execution time and to evaluate the speed of our package's implementation of AMKAT, which is written mostly in C++ and makes extensive use of the speed-optimized matrix operations in the C++ Armadillo library.

Algorithms for AMKAT

We begin by considering the algorithms for AMKAT with and without the PhiMr filter.

Without PhiMr

When testing without PhiMr, a description of the algorithm for AMKAT is as follows:

Algorithm 1: AMKAT (without PhiMr)

```
1. Adjust for covariates by fitting the null model using ordinary least squares
2. For the $j$th response variable, select a kernel as follows:
    a. For each candidate kernel:
        1. Compute the $n \times n$ empirical centralized kernel matrix
        2. Use the kernel matrix and the residuals/standard errors from step 1 to compute the statistic
           $\tau_j$ and its estimated asymptotic variance $\hat{\sigma}^2_{\tau_j}$
    b. Select the candidate kernel that maximizes $\tau_j/\hat{\sigma}^2_{\tau_j}$
3. Compute the test statistic $T$ using the kernels selected in step 2
4. For $b = 1, \dots, B$:
    a. Permute the row (observation) indices of the matrix of residuals from step 1
    b. Repeat steps 2-3 to obtain a permutation statistic $\ddot{T}_b$
5. Estimate a $P$-value as
```

$$\hat{P}_T = \frac{\left(\sum_{b=1}^{B} I(T \le \ddot{T}_b)\right) + 1}{B}$$

The computations involved in step 2.b and step 5 are trivial, while step 3 does not require any additional computation, as the test statistic is constructed by summing together quantities computed during step 2.

For M response variables, `amkat()` without PhiMr is roughly equivalent in computational expense to the following set of function calls:

- 1 call to `.fitAmkatNullModel()` (implements step 1)
- For each candidate kernel:
  - $M(B+1)$ calls to `generateKernelMatrix()` (implements step 2.a.1)
  - $M(B+1)$ calls to `.estimateSignalToNoiseRatio()` (implements step 2.a.2)

With PhiMr

When testing with the PhiMr filter, a description of the algorithm for AMKAT is as follows:

Algorithm 2: AMKAT with Testing Subset Selection Using PhiMr

```
1. Adjust for covariates by fitting the null model using ordinary least squares
2. Apply the PhiMr filter to select a testing subset $\mathcal{S} \subseteq \{1, \ldots, p\}$
3. For each dimension $j$ of $\boldsymbol{Y}$, select a kernel for $Y_j$ as follows:
   a. For each candidate kernel:
      • Compute the $n \times n$ empirical centralized kernel matrix using the testing subset $\mathcal{S}$
      • Use the kernel matrix and the residuals/standard errors from step 1 to compute the statistic
        $\tau_j$ and its estimated asymptotic variance $\hat{\sigma}^2_{\tau_j}$
   b. Select the candidate kernel that maximizes $\tau_j / \hat{\sigma}^2_{\tau_j}$
4. Compute the test statistic $T_{\mathcal{S}}$ using the kernels selected in step 2
5. Repeat steps 2-4 until $Q$ values of $T_{\mathcal{S}}$ have been generated; denote their sample mean as $\overline{T}_{\mathcal{S},Q}$
6. For $b = 1, \ldots, B$:
   a. Permute the row (observation) indices of the matrix of residuals from step 1
   b. Repeat steps 2-4 to obtain a permutation statistic $\ddot{T}_{\mathcal{S},b}$
7. Estimate a $P$-value as
```

$$\tilde{P}_{T_{\mathcal{S}},Q} = \frac{\left(\sum_{b=1}^{B} I(\overline{T}_{\mathcal{S},Q} \le \ddot{T}_{\mathcal{S},b})\right) + 1}{B}$$

For M response variables, `amkat()` with PhiMr is roughly equivalent in computational expense to the following set of function calls:

- 1 call to `.fitAmkatNullModel()` (implements step 1)
- $(B+Q)$ calls to `phimr()` (implements step 2)
- For each candidate kernel:
  - $M(B+Q)$ calls to `generateKernelMatrix()` (implements step 3.a.1)
  - $M(B+Q)$ calls to `.estimateSignalToNoiseRatio()` (implements step 3.a.2)

When compared to AMKAT without PhiMr, using PhiMr incurs the following differences:

- An additional $(B+Q)$ calls to `phimr()`
- For each candidate kernel:

  - The $M(B+1)$ calls to `generateKernelMatrix()` using the full $n \times p$ matrix $\mathbf{X}$ are now replaced by $M(Q+B)$ calls that each use some subset $\mathcal{S}$ of the columns in $\mathbf{X}$
  - An additional $M(Q-1)$ calls to `.estimateSignalToNoiseRatio()`

Note: Each call to `generateKernelMatrix()` will not use the full $n \times p$ matrix $\mathbf{X}$; rather, it will use only those columns specified by the testing subset $\mathcal{S}$.

Benchmarking

As noted in the previous section, the computation done by `amkat()` consists of calls to either three or four helper functions, depending on whether PhiMr is used.

To see how the execution time of each of these helper functions (as well as `amkat()` itself) scales with n and p, we simulated data for various values of (n, p) according to the design used in our power simulations and recorded the median execution time of each function over a large number of executions.

All code was executed on Windows 10 using a single core of a 2014 4th Generation (Devil's Canyon) Intel Core i5 4960K with a 6MB cache operating at a core clock frequency of 4.5GHz, a bus clock frequency of 100MHz and an uncore clock frequency of 3.5GHz, as measured by HWiNFO64v7.22-4731 (http://www.hwinfo.com).

The following simulation settings were used in all scenarios:

```
x_type <- 'cts' # continuous features
error_distribution <- 'normal' # multivariate normal errors
error_correlation_strength <- 0 # uncorrelated error components
signal_density <- 'sparse' # 7-variable signal set
```

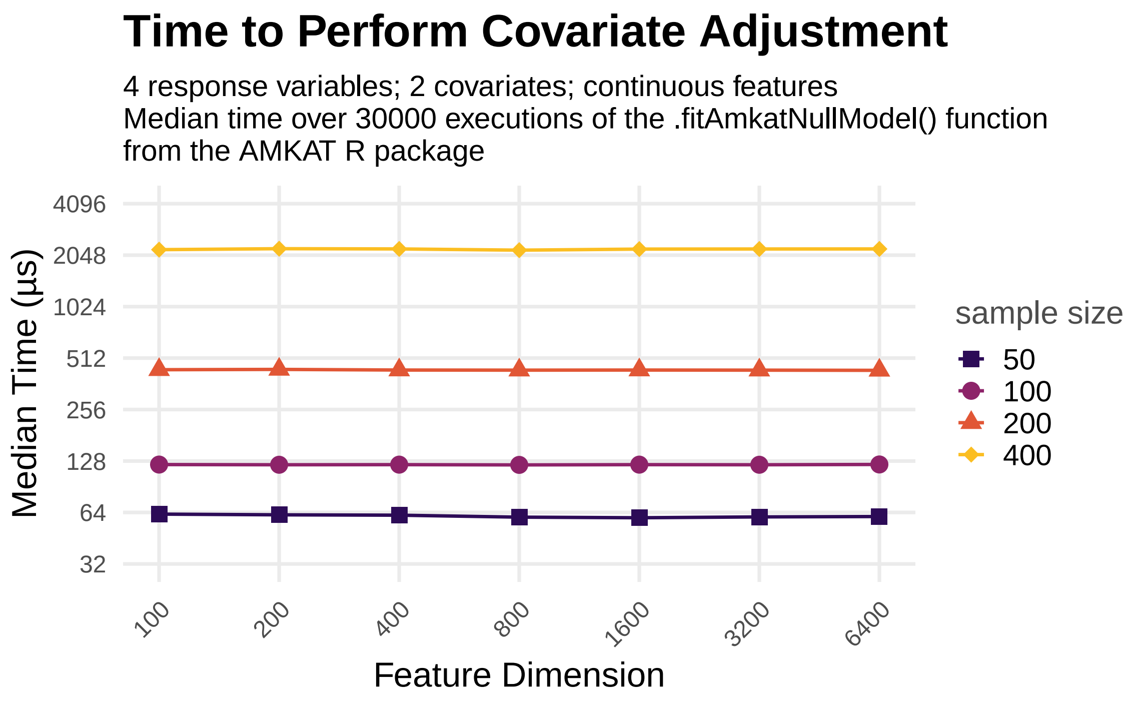
We initialize the workspace and load the data containing the simulation results:

```
# Prepare working environment
pkgs_to_load <- c('dplyr', 'ggplot2', 'tidyr', 'viridis', 'cowplot')
lapply(X = pkgs_to_load, FUN = library, character.only = TRUE)
dir_main <- dirname(dirname(rstudioapi::getActiveDocumentContext()$path))
dir_src <- file.path(dir_main, 'source_scripts')
source(file.path(dir_src, 'define_directories.R'))
source(file.path(dir_src, 'initialize_runtime_benchmarking.R'))
source(file.path(dir_src, "define_plot_functions.R"))
source(file.path(dir_src, "define_plot_settings.R"))
load(bench_plotdata_file)
```

Function to Perform Covariate Adjustment

We plot the results for the function `.fitAmkatNullModel()`, which computes the residuals and standard errors from the null model fit using ordinary least squares.

```
makeGenericBenchPlot(
  data_cov, reps_cov, 1e+3, "Median Time (\U00B5s)",
  "Time to Perform Covariate Adjustment",
  paste0(
    "4 response variables; 2 covariates; continuous features\nMedian time over ",
    reps_cov,
    " executions of the .fitAmkatNullModel() function\nfrom the AMKAT R package"),
  theme_settings = theme_bench_phimr(aspect = 0.5))
```



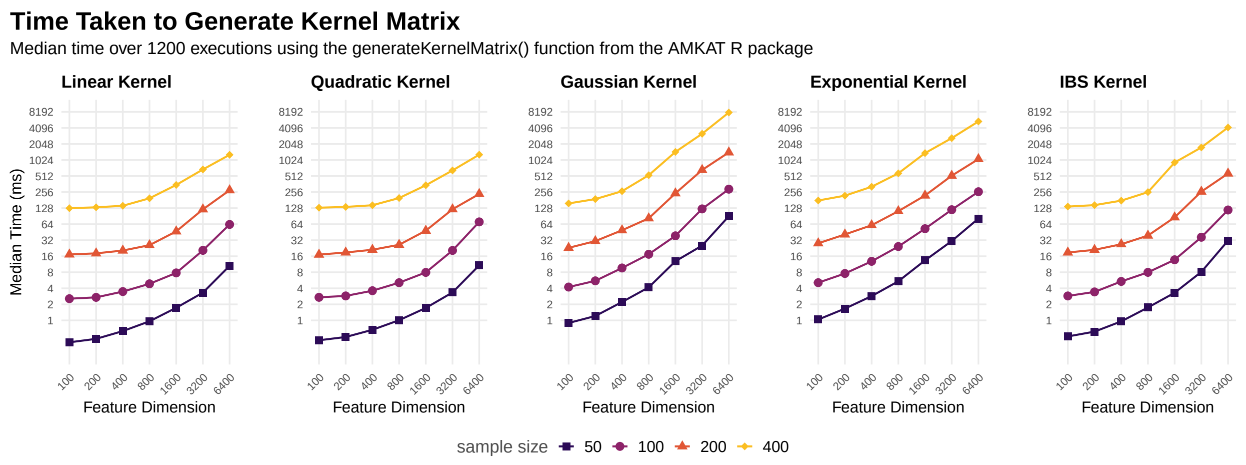
The function's execution time is constant in p, as $\boldsymbol{X}$ is absent from the null model.

Despite the observed rate at which this function's execution time grows with sample size, when we consider the ranges of execution times seen here, as well as the fact that this function is called only once by `amkat()`, we do not expect it to meaningfully affect the total execution time in a practical setting.

Function to Compute Kernel Matrix

We plot the results for the function `generateKernelMatrix()`, which computes the $n \times n$ empirical centralized kernel similarity matrix of a $n \times p$ matrix using a specified kernel function.

```
makeKernelBenchPlots(data_kermat, title_settings = title_bench_ker(reps_kermat))
```



The execution time with each kernel function exhibits faster-than-polynomial growth with respect to p: both axes are plotted on a $\log_2$ scale, so straight lines correspond to a polynomial rate of growth (with the slope corresponding to the degree of the polynomial). Despite the growth in execution time accelerating more rapidly than for polynomial growth, we note that over the range of values for p seen here, the instantaneous growth rate at each value of p was still catching up to that of a linear or quadratic function, depending on the kernel and sample size.

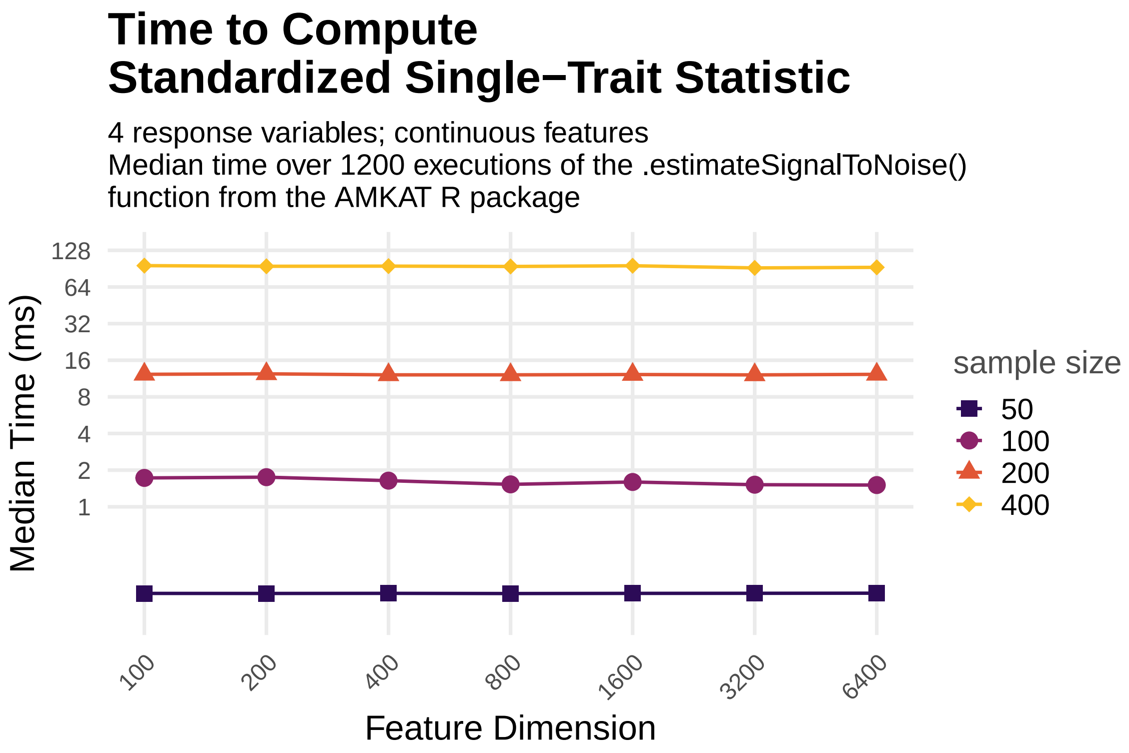
The growth rate in execution time with respect to sample size appears possibly faster-than-polynomial, though over the range of values considered here, the instantaneous rate of change was comparable to quadratic or cubic growth: Each time the sample size doubled, the execution time was scaled by a factor somewhere between 2^2 and 2^3.

We also note that the execution times for the linear and quadratic kernels were lower than for the other three kernels, and also appeared to grow more slowly.

Function to Compute τ_j and $\hat{\sigma}^2_{\tau_j}$

We plot the results for the function `.estimateSignalToNoise()`, which computes the statistic τ_j and its estimated asymptotic variance given inputs consisting of a kernel matrix, vector of residuals for Y_j and the corresponding standard error from the fitted null model.

```
makeGenericBenchPlot(
  data_stat, reps_stat, 1e+6, "Median Time (ms)",
  "Time to Compute\nStandardized Single-Trait Statistic",
  paste0(
    "4 response variables; continuous features\nMedian time over ",
    reps_stat,
    " executions of the .estimateSignalToNoise()\nfunction from the AMKAT R package"),
  theme_settings = theme_bench_phimr(aspect = 0.5))
```

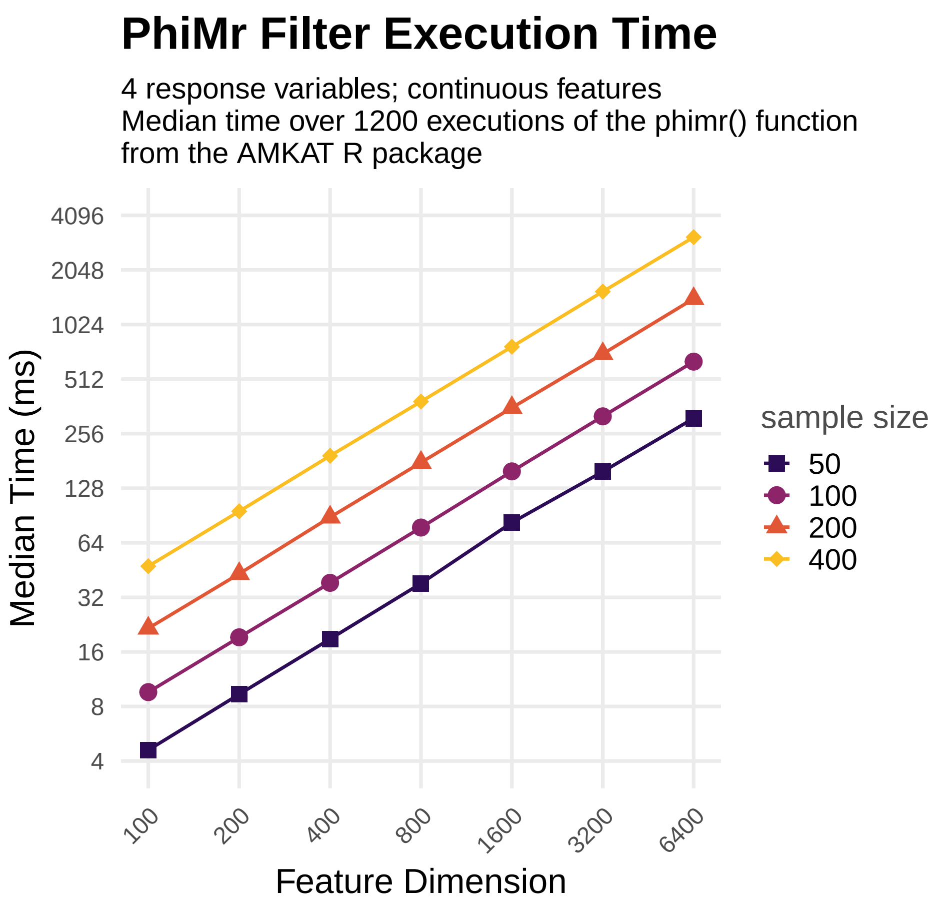


The bulk of the computation consists of three instances of multiplying two $n \times n$ matrices. Because the kernel matrix is computed outside of this function, p does not affect the scaling of execution time here.

The execution times are slightly below those for generating a kernel matrix at $p = 100$; thus, at low feature dimensions, similar shares of the total execution time for AMKAT will come from `.estimateSignalToNoise()` and `generateKernelMatrix()`, while at very high feature dimensions, the contribution from `generateKernelMatrix()` will dominate due to its scaling with respect to p.

PhiMr Filter

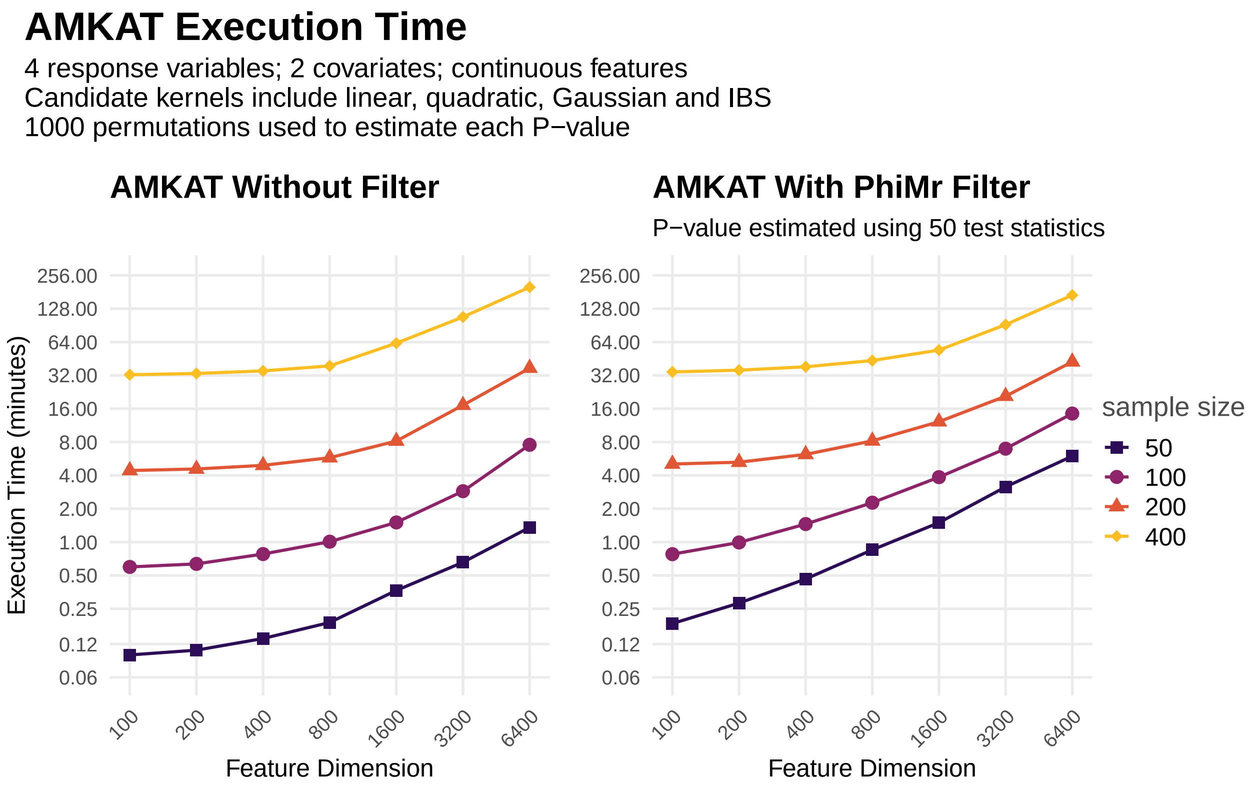
```
makePhimrBenchPlot(data_phimr, reps_phimr)
```



The execution time for PhiMr appears to grow close to linearly, with execution time doubling each time the sample size or feature dimension doubles. This is not surprising, given that for M response variables, PhiMr consists of Mp one-dimensional tests, each with sample size n.

AMKAT, Full Test

```
makeAmkatBenchPlots(
  data_amkat, title_settings = title_bench_amkat(num_permutations))
```



Looking at the range of execution times, we see that when the sample size and feature dimension are both large, our implementation of AMKAT can take multiple hours to run. Depending on the number of tests and availability of computing resources, this could pose potential challenges to practical use in certain settings. For small and moderate sample sizes, however, the execution was quite fast for an adaptive permutation-based test, in many cases taking only a few minutes or even seconds.

Comparing the scaling of execution time with PhiMr versus without PhiMr, initially we notice that using PhiMr often resulted in longer execution times, especially at smaller sample sizes.

When looking at $n = 400$, however, we notice that the execution time was nearly equal at $p = 100$, and grew more slowly in p with PhiMr than without PhiMr. We also notice that at all sample sizes, the execution times with PhiMr appear to exhibit slower acceleration with respect to p when compared to those with PhiMr.

What is going on here? Recall that compared to AMKAT without PhiMr, AMKAT with PhiMr involves the following differences in computation:

1. An additional $(B + Q)$ calls to `phimr()`
2. For each candidate kernel:
   a. The $M(B + 1)$ calls to `generateKernelMatrix()` using the full $n \times p$ matrix $\mathbf{X}$ are now replaced by $M(Q + B)$ calls that each use some subset $\mathcal{S}$ of the columns in $\mathbf{X}$
   b. An additional $M(Q - 1)$ calls to `.estimateSignalToNoiseRatio()`

As per (2.a), testing subset selection with PhiMr reduces the column dimension of each input matrix supplied to `generateKernelMatrix()`, a function which we observed to scale in faster-than-polynomial time with p. In our simulations to evaluate the PhiMr filter, we saw that for the continuous features used in the present simulation, PhiMr consistently removed half of the noise variables. The 7-variable signal set used here means that most of $\boldsymbol{X}$ is noise, and so we can reasonably expect PhiMr to have reduced feature dimension by half, on average, prior to each time `generateKernelMatrix()` was called. Thus, despite PhiMr requiring more calls to `generateKernelMatrix()`, the time taken for each was less than in the case without PhiMr, with this discrepancy widening as p increased.

Meanwhile, the computational cost of the $(B + Q)$ calls to `phimr()` scale at a roughly linear rate in both n and p, while the additional $M(Q - 1)$ calls to `.estimateSignalToNoiseRatio()` for each candidate kernel have a computational cost that is constant in p.

At small sample sizes, the cost of the calls to `phimr()` made up a greater share of AMKAT's total run time, since the execution time of `generateKernelMatrix()` was observed to grow at polynomial time or faster with respect to n, while that of `phimr()` grew linearly. This helps to explain why execution times for AMKAT with PhiMr at small sample sizes were longer than those without PhiMr across all values of p considered.

At greater sample sizes, when p was small, the additional cost of the calls to `phimr()`, `generateKernelMatrix()` and `.estimateSignalToNoiseRatio()` appear to have been offset by the reduction in costs from shrinking the dimension of the inputs to `generateKernelMatrix()`. As p grew, this reduction eventually began to outweigh the additional costs, resulting in AMKAT taking less time with PhiMr than without.

The implication here is that despite the procedure for AMKAT being more complex when testing with PhiMr, the additional complexity does not appear to have a practical impact on execution time (and may even have a beneficial impact for longer run times). Thus, so long as it can be expected to yield even marginal benefits and has no other notable downsides, testing with PhiMr can be recommended for use with AMKAT by default.