Memory Management for Large *N x T* Matrices

POLS 904 - Paul Johnson - FS17 - University of Kansas

Patrick John Gauding

December 22, 2017

This project demonstrates the feasibility of using sparse matrices to improve memory management and performance in R. Specifically, I modify and extend the capabilities of the `pcse` package by implementing sparse capabilities provided by the `Matrix` package. I identify a recurring issue in processing large $N\ x\ T$ matrices in which the package fails to process the required Kronecker multiplication and crashes R. Then, I identify the matrix storage structure used in the original package, which I hypothesize causes memory overload. I provide background on the sparse matrix alternative, and provide examples of implementation. I find notable improvements in both dimensional size and speed capabilities.

1 Panel-corrected Standard Errors and The Purpose of the `pcse` Package

The `pcse` package is an implementation of the panel-corrected standard errors technique proposed by Beck and Katz (1995; 2006). They argue that previous approaches to dealing with the inherent spatial and temporal correlations in time-series cross-sectional data, such as the Parks (1967)feasible generalized least squares (FGLS) approach, produce standard errors which are inherently incorrect. This is because of the assumption of GLS that the error generating process is known is impossible to fulfill in practice. The Parks approach assumes an AR1 autocorrelation and unit-specific serial correlation. Beck and Katz find these assumptions problematic, in that "the 'interesting' parameters of the model, β, do not vary across units; this assumption of pooling is at the heart of TSCS analysis." The correction model makes blunt assumptions about the characteristics of the error relationships, which may or may not exist in the data.

Beck and Katz instead propose using ordinary least squares with correction based on a block matrix of contemporaneous covariances. This model offers several advantages, primarily stemming from the spherical assumption of errors in OLS. These errors are consistent, and so provide a less biased (i.e. more correct) correction matrix. The upshot is that this technique is the error corrections are derived from the relationships of the errors themselves, instead of assuming relationships, as results from cases in which PCSEs are not necessary match OLS estimates (Beck and Katz, 1995, 641).

2 Identifying the Issue

The `pcse` package successfully estimates PCSEs for datasets of sizes practical for many applications in political science, particularly country-year data. However, the package breaks down and causes the R session to crash for larger datasets (greater than roughly *250 x 250*). Additionally, the package is impractically slow as the matrix becomes larger, taking roughly 2.75 minutes to calculate a near-maximum (*250 x 250*) successful run. My assumption is that the issue causing the package to both run slowly and to crash at higher levels of $N\ x\ T$ are related. Poor memory management could be expected to cause both the length of the calculation and the R session crashing. After running `debug()` on the `pcse` function, I identified the relevant section of code.

```
if (flag) {
  e <- using$resid
  E <- matrix(e, nCS, nTS, byrow = TRUE)
  E <- t(E)
  Sigma.hat <- crossprod(E)/nTS
  X <- as.matrix(using[, 4:dim(using)[2]])
  omega <- kronecker(Sigma.hat, diag(1, nTS))
  middle <- t(X) %*% omega %*% X
  nobs <- length(e)
  dataX <- X
}
```

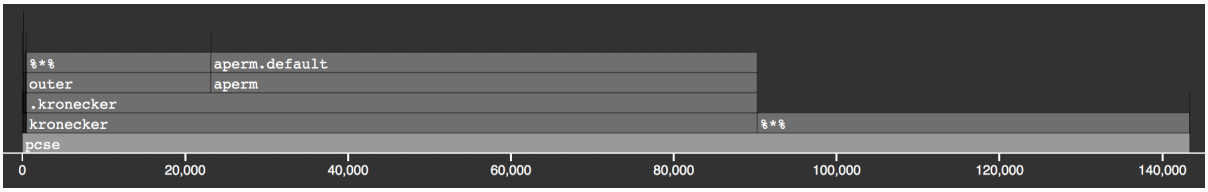

Figure 1: Graphical Representation of Memory Demand of `pcse` Function Calls

3	0	0	0	2
0	2	1	0	6
0	0	3	2	0
0	0	0	4	0
1	0	0	0	5

Figure 2: Full Matrix

3	.	.	.	2
.	2	1	.	6
.	.	3	2	.
.	.	.	4	.
1	.	.	.	5

Figure 3: Sparse Matrix

In this portion of the function, the matrix of summed covariances is calculated and then multiplied by the matrix of observations using a Kronecker product. The Kronecker product is a generalization of outer product of the two matrices. In other words, each of the observations is multiplied by each of the covariances, resulting in the ballooning in size of the matrix, to the point of slowing the entire function. Figure 1 is a graphical representation of the stacked memory demands of a run of the `pcse` function. Essentially, the only major memory demands in the function are the Kronecker operations, as well as the regular matrix multiplications. All of the other function calls are so insignificant by comparison as to not even show up.

3 Sparse Matrices

One approach to minimizing memory demand in large calculations is take advantage of the fact that many of values in a large matrix will be zero. A sparse matrix is useful for this task. Sparse matrices take a filled matrix and create an index of non-zero observations. The zeros are presumed to fill the rest of the matrix. Essentially, a sparse matrix is a map through which one can reconstruct the "true" matrix. Eliminating the zeros from being held in memory directly is advantageous as it should severely reduce the number of operations which need to be completed, as the zeros are no longer being multiplied.

A sparse matrix does not truly eliminate the zeros. Instead, they are removed and replaced with placeholder, represented in Figure 3 by periods. The zeros in question are in the covariance

matrix, which is function of the residual calculations performed by the initial `lm` call required before running the `pcse`function. Since a great number of the correlations of the errors will be zero, using a sparse matrix will reduce the number of actual calculations performed. I expect that by their elimination the function should perform much more quickly and at higher dimensions.

4 Implementation

To implement this approach, I return to the relevant section of the function displayed above. The goal is to treat the appropriate matrices before they enter the `kronecker` call. Sparse matrix capabilities are available in the `Matrix` package in `R`. The package includes a number of improvements on the base `R` package `matrix`. Implementing sparse matrices into this section is trivially easy.

```
  library(pcse)
library(Matrix)
if (flag) {
    e <- using$resid
    E <- Matrix(e, nCS, nTS, byrow = TRUE, sparse = TRUE)
    E <- t(E)
    Sigma.hat <- crossprod(E)/nTS
    X <- as.matrix(using[, 4:dim(using)[2]])
    omega <- kronecker(Sigma.hat, diag(1, nTS))
    middle <- t(X) %*% omega %*% X
    nobs <- length(e)
    dataX <- X
  }
```

The changes the code are subtle. I have simply capitalized the letter M in the previous `matrix` call, and added the argument `sparse = TRUE`. The only other changes are to load the `Matrix` and `pcse` libraries at the beginning of my `pcse_sparse` function. The `pcse` library loading is necessary for the summary function to correctly interpret the model output, but otherwise has no bearing on the revised function.

		Estimate	PCSE	t-value	Pr $> \|t\|$
Original					
	Intercept	0.032	0.096	0.329	0.742
	x	25.000	0.002	13036	0
Sparse					
	Intercept	0.032	0.096	0.329	0.742
	x	25.000	0.002	13036	0

Figure 4: Comparison of Estimations

5 Results

I then test the function on simulated data. I set five parameters manually (N, T, β, $\bar{y}$, and σ^2) and generate a random set of observations. I have confined this analysis to square matrices only (i.e. $N = T$), as my interest is in testing the dimensional limits of the function. To be clear, TSCS analysis will very rarely involve square matrices, so these dimension are not meant to simulate a practical analysis, but are rather a matter of convenience for repeated testing. For these tests, I have arbitrarily set $\beta = 25$, $\bar{y} = 50$, and $\sigma^2 = 5$. Because I am drawing from the same seed for these data, all estimates are equal regardless of N *x* T. For the sake of simplicity, I present this table only once in Figure 4.

The first issue to be resolved is whether the function runs and gives the same estimates as the original function. In all cases the estimates are identical. Having cleared that hurdle, the next immediate observation is the speed comparison between the two functions at dimensions `pcse` can handle. Using `benchmark` from the `rbenchmark` package, I simulate 1000 runs of both `pcse` and `pcse_sparse` at the *10 x 10* and *100 x 100* dimensions, and 10 runs at the *250 x 250* level. These results are presented in Figure 5.

These results show the benefit of using sparse matrices. Although the time for the smallest dataset slowed slightly, the improvements in larger datasets are significant. The *100 x 100* calculation is 18.247 times faster using `pcse_sparse` in comparison to `pcse`, saving more than 30 minutes. The difference is even more dramatic for the *250 x 250* calculation, with `pcse_sparse` being 89.083 times faster. To illustrate this difference visually, I run `profvis` on `pcse` and `pcse_sparse` back to back for the *250 x 250* dataset. In Figure the `pcse_sparse` results are barely visible in contrast

Test	*N x T*	Replications	Elapsed (min)	Relative
Original	10 x 10	1000	0.032	1
Sparse	10 x 10	1000	0.059	1.831
Original	100 x 100	1000	32.042	18.247
Sparse	100 x 100	1000	1.756	1
Original	250 x 250	10	26.533	89.083
Sparse	500 x 500	10	0.300	1

Figure 5: Benchmarking of 1000 Runs

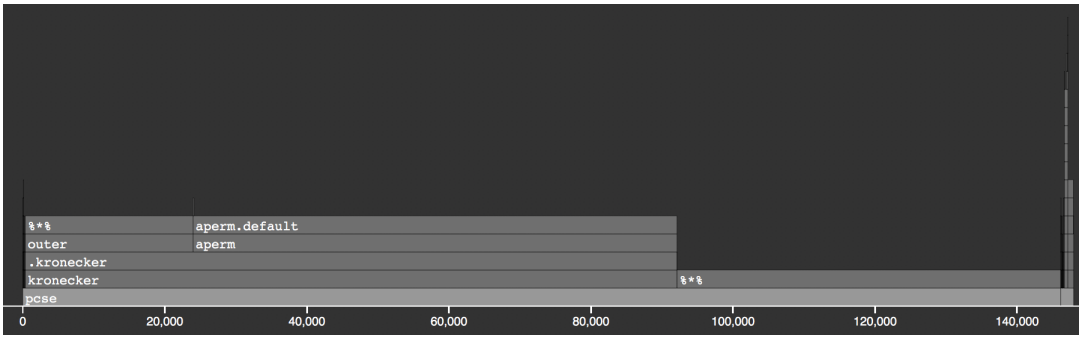

Figure 6: Profile of `pcse` and `pcse_sparse` for *250 x 250* Dataset

to the amount of time taken by `pcse`.

Finally, I test for the maximum square dimensional capacity for each function. These results are presented in Figure 7. I use blunt force repetition to drill down on the tipping point for each function. `pcse` failed at *257 x 257* and `pcse_sparse` failed at *834 x 834*. This represents a 1,053% increase in dimensional capacity.

Test	*N x T*	Elapsed (min)
Original	257 x 257	2.781
Sparse	834 x 834	6.20

Figure 7: Maximum Dimensional Capcity

6 Improvement and Extension Opportunities

These results demonstrate the benefits of improved memory management using sparse matrices. The ease and simplicity of this change, combined with the massive improvement in speed and dimensional capacity, lead me to suggest that the use of sparse matrices could be a best practice for a package involving large N matrix multiplication. This work demonstrates that little work is necessary to achieve these gains. However, I cannot be certain whether these findings generalize to other packages with similar issues, or if this is a special case.

This research has some limitations, leading to some opportunities for improvement. First, these tests were conducted on essentially the same dataset, set in square dimensions. This is not a realistic recreation of practical TSCS data, which generally has $T > N$ dimensions. A more thorough exploration of the differences between `pcse` and `pcse_sparse` would involve a Monte Carlo simulation to test dimensional flexibility. Are there any advantages or disadvantages to using sparse matrices in differing dimensions? Theoretically, I would expect that `pcse_sparse` would both improve speed and dimensional flexibility.

Ideally, another simulation would compare time and memory usage for all i from 1 to 257, the `pcse` maximum dimension. I attempted to utilize the `memtime` library, as well as `gc` to gather these data in a `for` loop, with the hopes of graphing the differences as dimension size increased. Due to time constraints, I am unable to present these results. My expectation is that both the time and memory demand for `pcse`will increase much more dramatically than `pcse_sparse`.

Finally, this research does not explore complications with any of the additional options available in pcse, particularly whether the data are balanced or unbalanced. I do not expect that there would be any important differences, as the use of sparse matrices seems to have had no impact other than memory management. However, it cannot be clear if implementing sparse matrices is a general improvement to the pcse function without testing.

Acknowledgements

I thank Clay Webb for the idea for this project, Paul Johnson for his patient and generous support, and Zack Roman for his help with the data generation code.

References

Beck, N. and Katz, J. N. (1995). What to do (and not to do) with time-series cross-section data. *American political science review*, 89(3):634–647.

Beck, N. and Katz, J. N. (2006). Random coefficient models for time-series–cross-section data: Monte carlo experiments. *Political Analysis*, 15(2):182–195.

Parks, R. W. (1967). Efficient estimation of a system of regression equations when disturbances are both serially and contemporaneously correlated. *Journal of the American Statistical Association*, 62(318):500–509.