# Appendix

## How to select the projecting subspace size () for MPOWIT?

If we set the multiplier to 1, regular subspace iteration is performed. As shown in Figure 9, subspace iteration takes a lot of iterations to converge and, therefore, is usually set to a value greater than 1. The number of iterations required for convergence decreases with an increase in the value of . However, a larger value of will increase the memory required to solve the PCA problem. We selected in all group-level PCA analyses using MPOWIT in this paper since this leads to a small memory burden in our rank PCA approximation. Depending on the computer memory resources available, a value of could be selected for very large group ICA studies to reduce the number of data loading operations when using the MPOWIT (un-stacked) approach.

|  |
| --- |
| Figure 9: **Finding the best block multiplier for MPOWIT**. The block multiplier is varied from 1 to 20 to determine the best multiplier which helps the MPOWIT algorithm to converge in fewer number of iterations without additional burden on memory. We used a large temporally concatenated fMRI dataset of size 66745 by 40000 and extracted 100 components in this experiment. |

## How much memory is required during the group-level PCA step?

Below, we give approximate memory requirements (in bytes) for each PCA algorithm. Let be the temporally stacked data as mentioned in the Materials and Methods section with dimensions , where is the number of subjects, is the number of components retained in the first PCA step, and is the number of voxels. We assume that the variables are stored in double precision (8 bytes) and that the first step PCA was completed. In the following we use to denote the number of components extracted in the second PCA step.

**EVD**:

Typically, a copy of the variable is stored in memory to compute , which is required to obtain the covariance matrix on the data. For large data analysis, in order to minimize the memory overhead of storing a copy of the variable in memory, only blocks of (one at a time) are stored and used to compute the covariance matrix in the smallest dimension of the data. Eigenvectors resulting from the eigen-decomposition are projected onto the data. The memory required in bytes is given by:

. (38)

If the voxel dimension is smaller than the stacked time dimension (Equation (5)), the memory required in bytes is given by:

. (39)

**Large PCA**:

Let be the number of block iterations and be the block length. The memory required by Large PCA for temporally stacked data is:

. (40)

If each subject’s data is loaded one at a time (i.e., un-stacked), the memory required is given by:

. (41)

**MPOWIT**:

Let be the multiplier used to define the subspace (block) size. The memory required by MPOWIT for temporally stacked data is:

(42)

The memory computation in Equation (42) is divided into three parts:

1. - Computation of the covariance matrix in the smallest dimension.
2. - Eigen-decomposition of the covariance matrix.
3. - is obtained by projecting onto the data: (this step is required for back-reconstruction using GICA1).

From Equation (42), the memory required for the un-stacked version of MPOWIT is:

. (43)

Note that when back-reconstruction is not performed is not required.

**SVP**:

Let be the number of components retained in the intermediate PCA step and be the subsampling depth selected. The memory required by SVP is given by:

. (44)

**STP**:

Let be the number of components selected in the intermediate PCA step and is the number of subjects in each group. The memory required by STP is given by:

. (45)

## Proof: MPOWIT and EM-PCA Converge to the Same PCA Subspace ()

Here we provide a proof that the algorithms presented for both MPOWIT and EM-PCA converge to the same PCA subspace estimate as long as both algorithms start from the same initial guess and execute the same number of iterations . The proof is presented using mathematical induction, initially showing the base cases for and iterations, and then proving that if it holds for some then it must also hold for iterations.

Preamble:

Assume is the singular value decomposition of . Thus,

If at iteration , is the -th orthonormal subspace estimate, then an approximation can be obtained as follows: project the data onto as ; from SVD of above and ; from EVD and ; then an approximation of the eigenvectors is . Thus, so that .

In the following, .

Proof by induction:

Initialize with Gaussian samples ():

A) Base cases:

:

MPOWIT EM-PCA

Thus, and ; and since is orthonormal, it follows that . Moreover, if all eigenvalues are distinct (like most cases in real data), it also follows that . When repeated eigenvalues are present, the EVD is not unique, particularly, the eigenvectors corresponding to repeated eigenvalues are not unique. However, practical EVD algorithms are deterministic in nature and consistently output the same set of eigenvectors. Therefore, so long as the EVD algorithm utilized for is the same as the one utilized for , then . For clarity, notice that for a subset of repeated eigenvalues, the trailing eigenvectors are immediately determined from the orthogonality condition after the first eigenvector of the subset is chosen. In other words, the entire set is determined by the choice of the first of the eigenvectors. Finally,

This concludes base case .

:

MPOWIT EM-PCA

Thus, and ; and since is orthonormal, it follows that . Moreover, based on the discussion for , it also follows that . Finally,

This concludes base case .

B) Inductive step:

MPOWIT EM-PCA

Thus, and ; and since is orthonormal, it follows that . Moreover, based on the discussion for , it also follows that . Finally,

Therefore, if both run for the same number of iterations and use the same initial guess .

## Parameter Selection for Large PCA

Here we study the optimality of Large PCA’s parameters and (see the section on Large PCA for a detailed description). We consider three different response measurements in our analysis: computation time in minutes, RAM memory used in GB, and number of dataloads. Since none of the three criteria alone suffices to establish preference of one parameter setting over the other, we considered all three simultaneously and determined the set of Pareto-optimal parameters based on the set of non-dominated points in the three-dimensional space of response measurements. The non-dominated points are those that cannot be outperformed simultaneously in all criteria by any other point. These are called Pareto-optimal and are indicated with a red circle in Figure 10 and Figure 11. The Pareto-optimal collection effectively outlines the trade-offs between each parameter setting. Points outside of the so-called Pareto-front are non-optimal since at least one method in the Pareto-front outperforms them in all criteria.

|  |
| --- |
| \\Loki\export\mialab\users\rsilva\projects\MPOWIT_PCA\LargePCAparam_pareto_1600.tif  Figure 10: Pareto-optimal settings for 1600 real fMRI datasets on a Linux server. The utopia point combines the best performance measurements across all Pareto-optimal points. |

|  |
| --- |
| \\Loki\export\mialab\users\rsilva\projects\MPOWIT_PCA\LargePCAparam_pareto_400.tif  Figure 11: Pareto-optimal settings for 400 real fMRI datasets on a Linux server. The utopia point combines the best performance measurements across all Pareto-optimal points. |

In practice, a parameter setting from the Pareto-front should be selected as a result of constraints from the problem at hand. For convenience, we select parameter setting L (, ) since it is Pareto-optimal regardless of the dataset size. Note that the times reported here reflect mostly the time spent on arithmetic operations because the caching functionality of the server operating system retained the datasets in memory between iterations of Large PCA. The memory usage we report does not include the operating system cache. Clearly, Large PCA should be tuned to the datatype and computing system available to attain optimal performance.