Let **A** be a SNP matrix

We define the Power Iteration below.

**Power Iteration**

**parameter:** A

k <- 0

xk <- N(0, 1)

**while** not converged:

x’k+1 = AATxk

xk+1, \_ = QR(x’k+1)

k = k + 1

**return** SVD(xk)

**Convergence:**

We measure convergence using two criteria.

1. Successive Ratio of Singular Values (“Q-Values”)
2. Root Mean Squared Error of top K EigenPairs (“RMSE\_K”)
   1. This is the metric that FlashPCA2 Uses

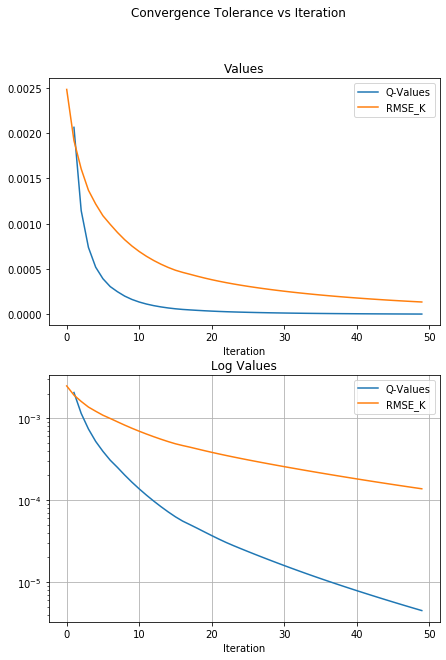
The relationship between Q-Values and RMSE\_K convergence is consistent and follows the trajectories seen in Figure 1a and Figure 1b. Both figures show the same run. However, figure 1b has an additional plot to show how the RMSE metric relates to the Q-Values metric. (The y scale was removed due to the distortion of squaring RMSE.)

The run in Figure 1a/b was on a 5K by 20K SNP artificially generated matrix with 5 dominate population sub groups. The eigenvalues are roughly as follows:

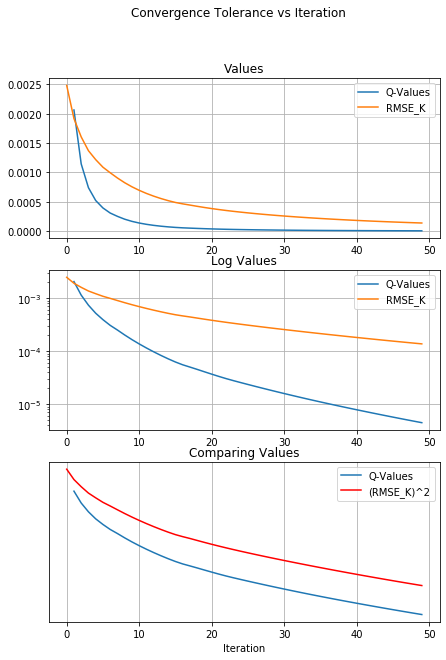
[74.90554143, 34.5336643 , 34.25206704, 34.14432925, 33.96509327,

11.41418892, 11.40732816, 11.39818389, 11.38732634, 11.37518494]

The Power Method was run for 50 iterations (0 -> 49) to find the top 10 Singular Values. A buffer size of 10 was used. A warm start was used that computed the initial V using 100 rows of the matrix. All computations were performed using 8Gbs of RAM and 4 Cores (A personal Laptop).

****

**Figure 1a: RMSE vs Q-Values**

****

**Figure 1b: RMSE vs Q-Values**

We can see that;

However, there are issues with both metrics when used to measure convergence of Growing Methods (to be discussed in following sections.)

**Large Matrix Testing:**

I ran the **Power Iteration** on two large matrices:

* 160K by 640K
  + 102.4 Billion Elements -> 102GBs in Dense Format 1 Byte Integers.
* 120K by 480K
  + 57.6 Billion Elements -> 50Gbs in Dense Format 1 Byte Integers

[NOTE] Standard float and integer in R is 8 Byte. Therefore, would take 8 times more space. While Python, C++, and other languages allow for 1 Byte integer types.

It is reasonable to expect the **Power Iteration** to take twice as long to complete a full iteration on a 160K by 640K matrix compared to 120K by 480K assuming similar resources.

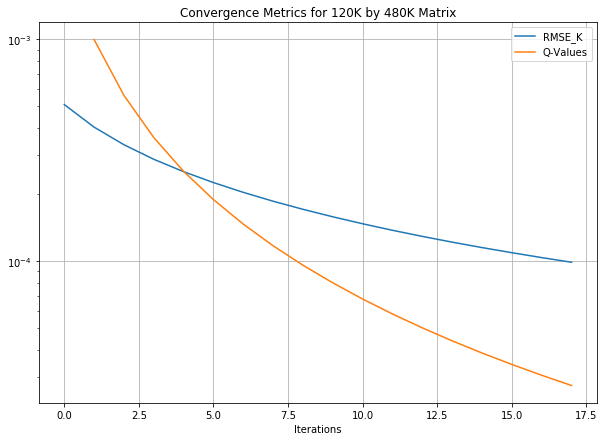
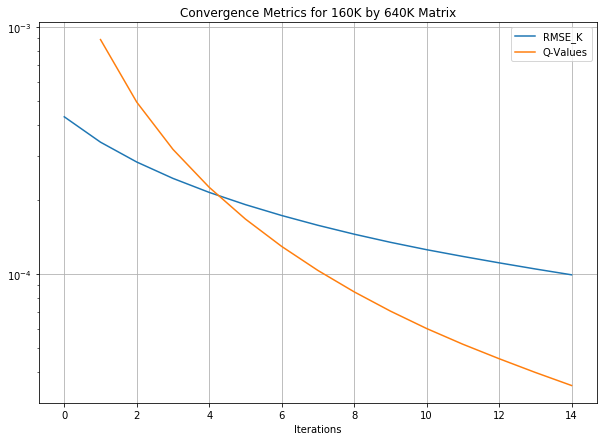
Test Parameters:

Top 10 Components (k = 10)

Buffer Size of 10 (b = 10)

Full Passes (Each iteration sees the full matrix and thus is not stochastic, p=1)

**Convergence Plots**

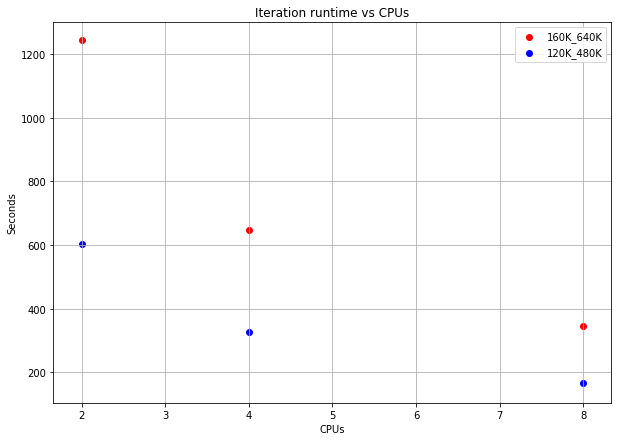


**Runtime Plots**

Runtime for a single iteration.

Allocated Memory was double the amount needed to load the matrix into memory

(To show effect of CPU and not Memory)



**Growing the Matrix:**

If we are seeking an approximation to the largest *K* singular values and respective vectors of a matrix, **A**, then purely from a parameter estimation perspective, a subset of the rows of **A** could be sufficient, if there is an underlying distribution for the rows of **A**.

In addition, one could ask how do the *K* singular values and respective vectors change as a matrix, **A**, change as new observations or features are added. One approach would be to simply create different sized submatrices of **A** and run the **Power Iteration** on each sub-matrix. However, this preforms repeated calculations that could be coupled for a more efficient algorithm.

**Notation**:

Let **I** be the set of row indices of **A**

Let **Ii** be the individual disjoint (pairwise) non-empty subsets of **I** whose union is **I.**

Let **f** be a fractional size of **I**i

**V** **Seed Power Iteration**

**parameter:** A

**parameter:** V

k <- 0

xk <- V

**while** not converged:

x’k+1 = AATxk

xk+1, \_ = TSQR(x’k+1)

k = k + 1

**return** SVD(xk)

**Successive Stochastic Power Iteration**

**parameter:** A

**parameter: 0 < f < 1**

create **I**1, **I**2, …, **I**c such that c = 1\f

V0 <- N(0, 1)

**for** i in {1, 2, …, c}:

Ai = A[{**I**1, **I**2, …, **I**i}, :]

Ui, Si, Vi = **V** **Seed Power Iteration(**Ai, Vi-1**)**

**If** converged:

**return** Ui, Si, Vi

**return** Uc, Sc, Vc

The **Successive Stochastic Power Iteration** can return the singular value decomposition of each submatrix A[{**I**1}, :], A[{**I**1, **I**2}, :], A[{**I**1, **I**2, **I**3}, :], … much faster than running each matrix through the **Power Iteration** separately.

In the following section we show the convergence and time analysis of the **SSPI** vs the **PI**.

All tests are on a 5K by 20K matrix.

Convergence results are on the top 10 singular values.

A Buffer size of 10 was used.

All computations were performed using 8Gbs of RAM and 4 Cores (A personal Laptop).

The **Power Iteration** was run for 50 iterations (0 -> 49) or until convergence, which ever came first.

For the Normal **Power Iteration**, A warm start was used that computed the initial V using 100 rows of the matrix

Using the same notation as before:

A1 = A[{I1}, :] (Uses 10% of the rows of A)

A2 = A[{I1, I2}, :] (Uses 20% of the rows of A)

…

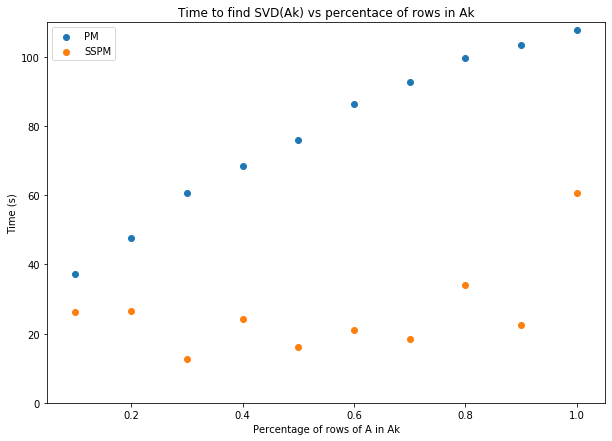
A10 = A[{I1, I2, …, I10}, :] (Uses 100% of the rows of A)

The **SSPI** method was run with 10 sections, f = .1, c=1/f = 10. This resulted in each sub-iteration, I, of the **V Seed Power Iteration** to run on Ai.

Therefore, **SSPI** ran SVD(A1) to convergence and used V1 to seed SVD(A2) which then seeded SVD(A3) and so on until SVD(A10) was computed.

The plot of the time of each run is below.

We can see that the PM is at best linear in the rows of Ai (This is because I limited the algorithm to 50 iterations). However, SSPM is more constant and quicker for each Ai.



Below is the plot of the cumulative time of

t1 + t2 + t3 + … + tk

and of

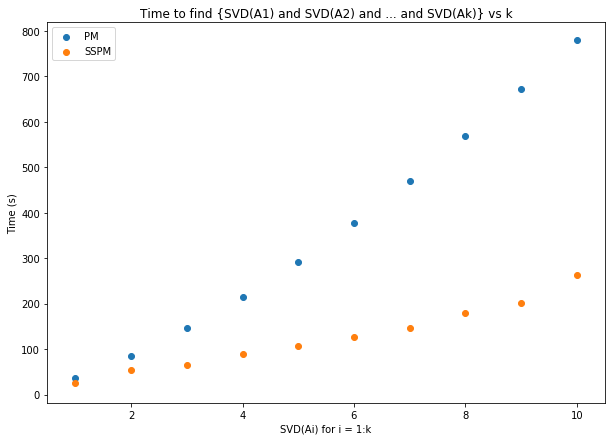
\tilde{t1} + \tilde{t2} + … + \tilde{tk}

Where;

ti = time for SVD(Ai)

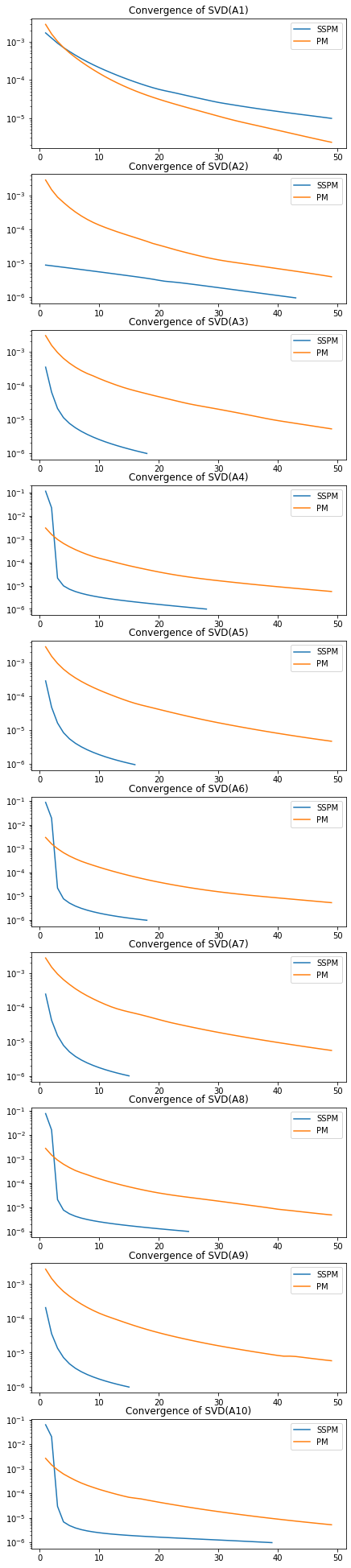
\tilde{ti} = time for SVD(Ai) using Vi-1 as seed.

We can see that PM scales quadratically and SSPM is much faster and scales roughly linearly.



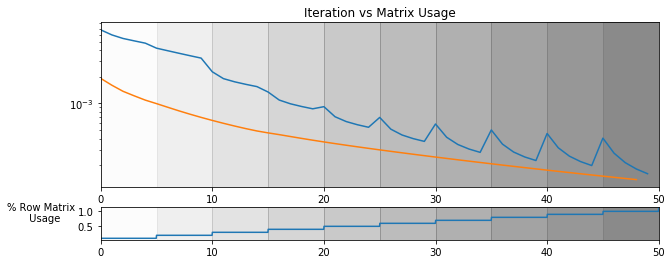
Finally, the convergence plots for all SVD(Ai). The first SVD(A1) for PM and SSPM takes roughly the same amount of time as they are both seeded from ~N(0,1). However, for the following SVDs on Ai, i > 1, **SSPI** finds convergence tolerance much faster due to the Vi-1 seed. Sometimes this takes a few iterations to adjust from Ai-1 to Ai, but this is quickly overcome.

We believe that this initial spike in tolerance during SVD(Ai) for i>1 is due to the metrics originally defined above which are not true accurate metrics for solution accuracy. We are developing a new subspace metric that should more accurately measure the convergence of the subspace.

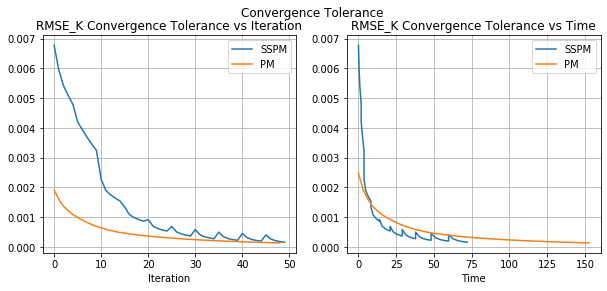


**Previous SSPI Analysis**

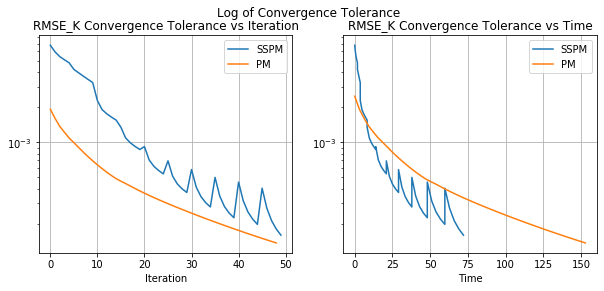
Comparing the convergence characteristics of the **Successive Stochastic Power Iteration** (SSPM/SSPI) vs the **Power Iteration** we see the relationship in the following charts below.



We can clearly see that the PM



**Figure 2a: SSPM vs PM Convergence**



**Figure 2b: SSPM vs PM Convergence**

The SSPM takes more iterations to converge primarily due to the large increase in error when the previous right singular vectors, **V**i-1, are used to seed the next **V Power Iteration**. The explanation for this error increasing so drastically is still unknown. It is worth noticing that the log increase in error when using **V**i-1 to seed the ith iteration of the **V Power Iteration** increases as the overall solution accuracy increases. (This is the increasingly large jump in the RMSE metric every time the array is “grown”). This argues against the idea that the SVD parameters can be learned with a fraction of the observations. However, if this increase in error each time the matrix is “grown” could be decreased then it could be possible to use a small fraction of the matrix to find a great solution for the full matrix. Then when growing the matrix the solution accuracy would not deteriorate as much.

However, the first iterations take much less time than the later iterations which can be seen in the graphs on the right side of the figure. The initial iterations when only 1/10th of the data is being used in the power iterations are almost 10 times faster than the full iterations.

The point of this experiment is to show:

1. Suppose you want to get SVD solutions for a sequence of matrices B\_{n \times p} [n = samples, p=features] for different values of p (p increases p1 < p2< p3 <…).
2. Then warm-starting SVDs as we increase p-values costs less than not warm-starting.

I believe that we are scaling up n, [n = samples], for different values of n (n increases n1 < n2 < n3 < …) to find how many samples we need to get a good estimate on V before diminishing returns “start/kick-in”. The dimensions of V are {k \times p} so V should be “blind” to [n = samples].

Warm starting seems to be beneficial. A lot of the information in the right subspace can be captured very efficiently with smaller subsets of the rows of B. (Similarly, this would work for the left hand subspace and subsets of the columns of B).

However, V found from SVD([B1; B2]) seems to have some significant “issue” for warm starting the SVD([B1; B2; B3]). (This is my explanation for the large jump in RMSE error every time the matrix is “grown”. For example, from [B1; B2] -> [B1; B2; B3].