I. Introduction

Java Essential Dynamics is a set of java programs for analyzing protein trajectories. The trajectories may be derived from MD, FIRST/FRODA, or other dynamic simulations that output a trajectory as a set of PDB files. The program can handle single chain PDB files with no chain identifier as well as multi chain PDB files that use chain IDs. The user may specify the set of residues to be considered for the analysis, and this set need not be continuous. As a pre-processing step, JED aligns all the structures in the trajectory to a specified reference structure using a quaternion alignment algorithm. The **conformation rmsd** is determined for each structure using a specified reference structure and the **residue rmsd** is determined from the entire trajectory. An **edited PDB file** is generated for the specified subset in which the B-factors have been replaced with the residue rmsd values for visualization purposes.

The core element of essential dynamics is performing principle component analysis (PCA) and JED implements two variations of PCA. The first and most common method is a Cartesian coordinate based analysis (cPCA)using the selected set of the alpha carbons. The second method is an internal coordinate based analysis using the distances (dPCA) between each residue and all the others in the protein. This is a computationally expensive algorithm and the interpretation is difficult for sets of residues greater than 10 (note that dPCA on a ten residue subset will yield eigenvectors having n*(n-1)/2 = 45 components, each corresponding to one set of inter-residue distances). The user can specify the number of Cartesian and distance modes to determine, as well as the number of Cartesian modes to visualize. Mode visualization is done by creating a set of 20 PDB files that capture the displacement of the alpha carbons for the given mode. A Pymol© script is generated to animate the frames. Note that dPCA modes cannot be mapped to the structure in an intuitive way.

The two methods of PCA are implemented using both the covariance matrix (Q) and the correlation matrix (R), as these two methods yield somewhat different results due to the inherent statistical biases in each approach. The program computes the **PCA modes** (RMSD and MSD) from the Cartesian eigenvectors so that they may be mapped to the residue set. Additionally, sets of structures can be generated to visually inspect the cPCA modes. Eigenvectors from the distance PCA cannot be mapped to the residue set in any simple way, so no mapping or visualization is attempted.

A set of **displacement vectors** (DV) is calculated using a specified reference structure and then those DVs are projected onto a set of eigenvectors (these are the principle components (PCs)). These **DV projections** (DVP), also known as the principle components (PCs) allow the user to investigate how the trajectory projects into the space defined by each eigenvector. The DVPs are given using un-normalized and normalized inner products as well as weighted by the corresponding eigenvalue.

JED performs a simple subspace analysis on the two sets of eigenvectors generated from the Q and R variants of PCA. This allows one to determine how different the PCA results are due only to the choice of PCA model. A comparison is also made to a random set of eigenvectors derived from the same vector space. This allows the user to establish a baseline for statistically valid subspace comparisons. Additional analysis can be done using the Subspace_Analysis.java class and its driver programs. To perform these kinds of tests, one first needs to generate sets of eigenvectors from each trajectory of interest as well as from the pooled trajectories to use as a reference set. Subspace analysis is done by comparing the sets of eigenvectors and determining the RMSIP,

Principal Angles (PA), cumulative overlap (CO), cosine products (CP), vectorial angular sum (VAS), and the maximum angle between subspaces of the given vector space.

II. Running the Program

JED is a java based set of programs and as such can run on any platform with a suitable JDK.

The program can be run from compiled source or from the provided executable jar files.

It is critical that the environment variable Java **CLASSPATH** be correctly set.

There are two driver programs for JED: One (JED_Driver) runs a single job using parameters specified in the input file, and the other (JED_Batch_Driver) runs a batch of jobs **sequentially**. The first is suited for running a single job at the command line or when using submit scripts on computer cluster resources. This can be implemented using job arrays so that your jobs run in parallel rather than sequentially.

Since JED requires an input file for job parameters, the run command takes one argument: the path to the input file. If no argument is specified, then JED assumes that the input file is named JED.txt and is located in the same directory from which the JVM was called.

To run JED at a command prompt or in a PBS script, you can use one of the following commands:

```
Java -d 64 JED_Driver "/path/to/your/input/file.txt" (This runs the compiled java program)

Java -jar -d 64 JED_Driver.jar "/path/to/your/input/file.txt" (This runs the executable jar file)

Java -d 64 JED_Batch_Driver "/path/to/your/input/file.txt" (This runs the compiled java program)

Java -jar -d 64 JED_Batch_Driver.jar "/path/to/your/input/file.txt" (This runs the executable jar
file)
```

Note: It is imperative that your CLASSPATH is correctly set. Alternatively, you can always specify the complete absolute path for files and and/or add the -cp option to the java command.

The Cobra computer cluster has a 64 bit JDK installed. Be sure to request adequate memory resources as JED computes the complete covariance matrix and holds it in memory. Additionally, matrix diagonalization scales as O^3 so depending on the size of your protein, you may need 12-36GB . Use BigMem if necessary (1TB).

Each job should be assigned to its own directory, which should contain either the PDB files to read or the coordinate file to process, along with the reference PDB file and residue lists for specifying the Cartesian and distance subsets.

NOTE: A preliminary run with no PCA should be run using all residues in the protein to generate a coordinate file for all the alpha carbons. This makes multiple subset analyses much more efficient as the program only needs to read in the PDB files once. After this step, the pdb files can either be deleted or archived. Alternatively, you can begin with a matrix containing the alpha carbon coordinates for all your frames. The matrix packing is as follows: Rows are variables and columns are frames. For N residues, there are 3N rows: N x coordinates, N y coordinates, and N z coordinates, stacked in that order.

The files that specify the residue lists for the Cartesian and Distance subsets are formatted as follows:

Single chain PDBs use a single column: Residue Number

Multi-Chain PDBs use a double column (tab separated): Chain ID, Residue Number

The main JED programs are called **JED_Driver.java** and **JED_Driver.jar** and their batch counterparts**.**

These programs obtain all run-time information from an input file that is read during execution:

The JED_Driver input file is named **JED.txt** by default.

The JED_Batch_Driver input file is named JED_Batch.txt by default.

It is critical that this file be constructed correctly and be located in the directory that you specify.

(Alternatively, if no path argument is passed, then where the JVM is called.)

The file is white space delimited .

The file will have different number of lines/tokens depending on the options selected:

The key differentiating factors are:

Will the program will read in all the PDB files in the working directory or read in a coordinate file?

Are the PDB files single chain or multi chain?

Will the analysis do cPCA, dPCA, BOTH, or NONE?

Will Cartesian mode visualization be done?

Output Files:

JED will create a directory named JED_results_$description under the working directory for its output files.

Cartesian PCA results will be in a sub directory names cPCA

Distance PCA results will be in a sub directory called dPCA.

If cPCA or dPCA exist, they will contain a sub directory named SSA in which the subspace analysis for the covariance versus correlation analysis.

The naming of the output files includes the ss_$number_of_residues header so that you will know how many residues the subset contained for the given results.

In the following examples, code and text file content is shown in `10 point Consolas`

EXAMPLE: PRELIMINARY RUN (for Single Chain PDB files, Single job driver):

This job will read PDB files, but will perform NO PCA.

The main purpose of this is to generate the matrix of alpha carbon coordinates.

For this to work, the residue list MUST contain ALL the residues specified in the PDB files.

This step should be done PRIOR to any other jobs as it will output ALL the alpha carbon coordinates.

```
-------------------------------------------------------------------------------
0.00
3.00
1     0
0     0     0
/dir/to/Cartesian/residue/list/  all_residues.txt    0
/working/directory/for/job/      Description1       reference_PDB_file1.pdb
-------------------------------------------------------------------------------
```

Notes:
Line #1 specifies the **percentage** of frames to remove based on conformation RMSD (float [0,1]) → 0.00=none
Line#2 specifies the **Z-score threshold** for outlier determination (float) → z=3 adjusts <1% of the samples
Line #3 – Token #1 specifies whether to **read PDB** files (0 or 1) →1=yes
Line #3 – Token #2 specifies if the PDB files are **Multi Chain** (0 or 1) →0=no
Line #4 – Token #1 specifies the **number of Cartesian modes** to process (integer) →0=none
Line #4 – Token #2 specifies the **number of Distance modes** to process (integer) →0=none
Line #4 – Token #3 specifies the **number of Cartesian modes to Visualize** (integer) →0=none
Line #5 – Token #1 specifies **the path to the Cartesian Subset Residue List** (String)
Line #5 – Token #2 specifies the **name of the Cartesian Subset Residue List** (String) → Specify ALL residues
Line #5 – Token #3 specifies the **chain offset** (integer)
Line #6 – Token #1 specifies the **working directory**(WD) for job (String)
Line #6 – Token #2 specifies the **description** for job (String)
Line #6 – Token #3 specifies the **reference PDB** file for job (String)

Output Files:
Key output files for this pre-process analysis are:
JED Log and PDB Read Log, **Original and Transformed PDB coordinates**, conformation and residue rmsds, and All_Residues_JED.txt , which is a file containing a list of all the residues that JED found in your PDB files.

As a "Best Practice", you should always do the PRELININARY RUN prior to any other analysis.
This means that you will only have to read the PDB files once (You can archive them or delete them)
It is absolutely critical that you include ALL the residues in the PDB files so that additional analyses will have access to any residue that you might specify. Each successive analysis will read the alpha carbon coordinates from the PDB coordinates file.

EXAMPLE: PRELIMINARY RUN (for Single Chain PDB files, Batch job driver):

The batch consists of 2 jobs , will read PDB files, and will perform NO PCA.

The main purpose of this is to generate the matrix of alpha carbon coordinates.

For this to work, the residue list MUST contain ALL the residues specified in the PDB files.

```
-------------------------------------------------------------------------------------
2
0.00
3.00
1     0
0     0     0
/dir/to/Cartesian/residue/list/        all_residues.txt          0
*************************************************************************************
/working/directory/for/job/1/          Description1              reference_PDB_file1.pdb
/working/directory/for/job/2/          Description2              reference_PDB_file2.pdb
-------------------------------------------------------------------------------------
```

Notes:
Line #1 specifies the number of jobs (integer) →2
Line #2 specifies the **percentage** of frames to remove based on conformation RMSD (float [0,1]) → 0.00=none
Line#3 specifies the **Z-score threshold** for outlier determination (float) → z=3 adjusts <1% of the samples
Line #4 Token #1 specifies whether to read PDB files (0 or 1) →1=yes
Line #4 Token #2 specifies if the PDB files are Multi Chain (0 or 1) →0=no
Line #5 Token #1 specifies the number of Cartesian modes to process (integer) →0=none
Line #5 Token #2 specifies the number of Distance modes to process (integer) →0=none
Line #5 Token #3 specifies the number of Cartesian modes to Visualize (integer) →0=none
Line #6 Token #1 specifies the path to the Cartesian Subset Residue List (String)
Line #6 Token #2 specifies the name of the Cartesian Subset Residue List (String) → Specify ALL residues
Line #6 Token #3 specifies the chain offset (integer)
Line #7 is a separator line: header/batch info above, and job listings below.
Line #8 Token #1 specifies the working directory(WD) for job 1 (String)
Line #8 Token #2 specifies the description for job 1 (String)
Line #8 Token #3 specifies the reference PDB file for job 1 (String)
Line #9 Token #1 specifies the working directory(WD) for job 2 (String)
Line #9 Token #2 specifies the description for job 2 (String)
Line #9 Token #3 specifies the reference PDB file for job 2 (String)

Output Files:
Key output files for this pre-process analysis are:
JED Log and PDB Read Log, **Original and Transformed PDB coordinates**, conformation and residue rmsds, and All_Residues_JED.txt , which is a file containing a list of all the residues that JED found in your PDB files.

As a "Best Practice", you should always do the PRELININARY RUN prior to any other analysis.
This means that you will only have to read the PDB files once (You can archive them or delete them)
It is absolutely critical that you include ALL the residues in the PDB files so that additional analyses will have access to any residue that you might specify. Each successive analysis will read the alpha carbon coordinates from the PDB coordinates file.

EXAMPLE: cPCA (for Single Chain PDB files using Single job driver)

The top 20 modes will be processed but no mode visualization will be done.

```
-------------------------------------------------------------------------------------------
0.00
3.00
0     0
20     0     0
/dir/to/Cartesian/residue/list/        subset_residues.txt       0
/working/directory/for/job/            Description               reference_PDB_file.pdb
PDB_coordinates.txt                    0
-------------------------------------------------------------------------------------------
```

Notes:

Line #1 specifies the **percentage** of frames to remove based on conformation RMSD (float [0,1]) → 0.00=none
Line#2 specifies the **Z-score threshold** for outlier determination (float) → z=3 adjusts <1% of the samples
Line #3 Token #1 specifies whether to read PDB files (0 or 1) →0=no
Line #3 Token #2 specifies if the PDB files are Multi Chain (0 or 1) →0=no
Line #4 Token #1 specifies the number of Cartesian modes to process (integer) →20
Line #4 Token #2 specifies the number of Distance modes to process (integer) →0=none
Line #4 Token #3 specifies the number of Cartesian modes to Visualize (integer) →0=none
Line #5 Token #1 specifies the path to the Cartesian Subset Residue List (String)
Line #5 Token #2 specifies the name of the Cartesian Subset Residue List (String) → Specify subset
Line #5 Token #3 specifies the chain offset (integer)
Line #6 Token #1 specifies the working directory(WD) for the job (String)
Line #6 Token #2 specifies the description for the job (String)
Line #6 Token #3 specifies the reference PDB file for the job (String)
Line #7 Token #1 specifies the name of the coordinates file for the job (String)
Line #7 Token #2 specifies the reference column in the coordinates matrix for the job (integer)

NOTICE! When reading from a coordinate file, TWO lines are necessary to specify a job.

Gotchas! (things that will generally make your life miserable…)

Dumb mistakes:

Does the input-file start on the first line? If not, → CRASH!!!
Is dPCA specified, but no residue list is specified for the distance subset? If so, → CRASH!!!
The WD contains PDB files of different sizes → CRASH!!!
The WD does not contain the reference PDB file → CRASH!!!
Is the integer an really an integer or is it a float or a string? → CRASH!!!
Does the directory string end in "/" or "\\"?: If not → CRASH!!!

More subtle mistakes:

Number of cPCA residues = N, If number of cPCA modes > 3N → CRASH!!!
Number of dPCA residues = N, If number of dPCA modes > N(N-1)/2 → CRASH!!!
Number of cPCA modes = 1, modes to visualize = 3 → CRASH!!!
The WD contains PDB files with 2 chains but no chain IDs → Non-sense results!
If the PDB file numbering starts at 5, then the chain offset is not 0, it is 4 → Garbage out!

EXAMPLE: Both cPCA & dPCA with visualization (for Single Chain PDB files, Single job driver):

The top 20 cPCA modes, and the top 3 dPCA modes will be generated;

The top 5 cPCA modes will processed for mode visualization.

```
-----------------------------------------------------------------------------------------
0.00
3.00
0     0
20     3     5     1.0
/dir/to/Cartesian/residue/list/  residue_list_cart_filename.txt   0
/dir/to/distances/residue/list/  residue_list_dist_filename.txt   0
/working/directory/for/job/    Description     reference_PDB_file.pdb
PDB_coordinates.txt       0
-----------------------------------------------------------------------------------------
```

Notes:
Line #1 specifies the **percentage** of frames to remove based on conformation RMSD (float [0,1]) → 0.00=none
Line#2 specifies the **Z-score threshold** for outlier determination (float) → z=3 adjusts <1% of the samples
Line #3 Token #1 specifies whether to read PDB files (0 or 1),
Line #3 Token #2 specifies if the PDB files are Multi Chain (0 or 1);
Line #4 Token #1 specifies the number of Cartesian modes to process (integer)
Line #4 Token #2 specifies the number of Distance modes to process (integer)
Line #4 Token #3 specifies the number of Cartesian modes to Visualize (integer) → 1.0 = default
Line #4 Token #4 specifies the scale factor for the visualization (double)
Line #5 Token #1 specifies the path to the Cartesian Subset Residue List (String)
Line #5 Token #2 specifies the name of the Cartesian Subset Residue List (String)
Line #5 Token #3 specifies the chain offset (integer)
Line #6 Token #1 specifies the path to the Distance Subset Residue List (String)
Line #6 Token #2 specifies the name of the Distance Subset Residue List (String)
Line #6 Token #3 specifies the chain offset (integer)
Line #7 Token #1 specifies the working directory(WD) for the job (String)
Line #7 Token #2 specifies the description for the job (String)
Line #7 Token #3 specifies the reference PDB file for the job (String)
Line #8 Token #1 specifies the name of the coordinates file for job 1 (String)
Line #8 Token #2 specifies the reference column in the coordinates matrix for the job (integer)

Remember! When reading from a coordinate file, TWO lines are necessary to specify a job.

IN SUMMARY:
IF ANY directory cannot be found or ANY file cannot be stat, the program crashes.
IF there is any problem with any input, the program crashes.
IF you specify impossible values, the program crashes.
The program was designed to crash rather than "catch" mistakes and output garbage.

Next, we consider how to construct the input file for multi chain PDB files:

EXAMPLE: THE PRELIMINARY RUN (for Multi Chain PDB file, Single job driver):

For this to work, the residue list MUST contain ALL the residues specified in the PDB files.

```
--------------------------------------------------------------------------------
0.00
3.00
1     1     2     A     B     795     151     0     0
0     0     0
/dir/to/Cartesian/residue/list/        all_residues.txt
/working/directory/for/job/            Description        reference_PDB_file.pdb
--------------------------------------------------------------------------------
```

Notes:
Line #1 specifies the **percentage** of frames to remove based on conformation RMSD (float [0,1]) → 0.00=none
Line#2 specifies the **Z-score threshold** for outlier determination (float) → z=3 adjusts <1% of the samples
Line #3 Token #1 specifies whether to read PDB files (1),
Line #3 Token #2 specifies if the PDB files are Multi Chain (1);
Line #3 Token #3 specifies the number of chains (2);
Line #3 Token #4 specifies the first chain ID (A);
Line #3 Token #5 specifies the second chain ID (B);
Line #3 Token #6 specifies the number of residues in chain A (795);
Line #3 Token #7 specifies the number of residues in chain A (151);
Line #3 Token #8 specifies the offset of Chain A (0);
Line #3 Token #9 specifies the offset of Chain B (0);
Line #4 Token #1 specifies no Cartesian modes to process (0)
Line #4 Token #2 specifies no Distance modes to process (0)
Line #4 Token #3 specifies no Cartesian modes to Visualize (0)
Line #5 Token #1 specifies the path to the Cartesian Subset Residue List (String)
Line #5 Token #2 specifies the name of the Cartesian Subset Residue List (All residues in PDB file)
Line #6 Token #1 specifies the working directory(WD) for the job (String)
Line #6 Token #2 specifies the description for the job (String)
Line #6 Token #3 specifies the reference PDB file for the job (String)

Another Gotcha…

Make sure that the chain information (IDs, lengths, offsets) is correct or the output will be non-sense!

Output Files:

Key output files for this pre-process analysis are:

JED Log and PDB Read Log, **Original and Transformed PDB coordinates**, conformation and residue rmsds, and All_Residues_Multi_JED.txt , which is a file containing a list of all the residues that JED found in your PDB files (two column format as required for input).

EXAMPLE: cPCA (for Multi Chain PDB files, Single job driver):

The top 20 modes will be processed, but no mode visualization will be done.

```
-------------------------------------------------------------------------------------------
0.00
3.00
0      1     2     A     B     795     151      0      0
20     0     0
/dir/to/Cartesian/residue/list/        subset_residues.txt
/working/directory/for/job/            Description              reference_PDB_file.pdb
PDB_coordinates.txt             0
-------------------------------------------------------------------------------------------
```

Notes:
Line #1 specifies the **percentage** of frames to remove based on conformation RMSD (float [0,1]) → 0.00=none
Line#2 specifies the **Z-score threshold** for outlier determination (float) → z=3 adjusts <1% of the samples
Line #3 Token #1 specifies whether to read PDB files (0 or 1) →0=no
Line #3 Token #2 specifies if the PDB files are Multi Chain (0 or 1) →1=yes
Line #4 Token #1 specifies the number of Cartesian modes to process (integer) →20
Line #4 Token #2 specifies the number of Distance modes to process (integer) →0=none
Line #4 Token #3 specifies the number of Cartesian modes to Visualize (integer) →0=none
Line #5 Token #1 specifies the path to the Cartesian Subset Residue List (String) → directory
Line #5 Token #2 specifies the name of the Cartesian Subset Residue List (String) → file name
Line #6 Token #1 specifies the working directory(WD) for the job (String)
Line #6 Token #2 specifies the description for the job (String)
Line #6 Token #3 specifies the reference PDB file for the job (String)
Line #7 Token #1 specifies the name of the coordinates file for the job (String)
Line #7 Token #2 specifies the reference column in the coordinates matrix for the job (integer)

Same gotchas apply here…make sure the files and paths exist!

Remember the format of the residue list:

TWO columns for Multi Chain Analysis: Chain ID, residue number

ONE column for Single Chain Analysis: residue number.

TWO VERY IMPORTANT FACTS:

If the number of cPCA modes > 0, then there must be a Cartesian residue list file specified!

If the number of dPCA modes > 0, then there must be a Distance residue list file specified!

III. JED Output

JED creates lots of output files. To keep things organized, JED creates a set of directories for all output data. The top level output is placed in a directory called "**JED_results_DESCRIPTION**", where DESCRIPTION is the description that you provided for the job in JED.txt. This directory is created during program execution and contains all the outputs from the program except the results of the subspace analysis and the mode visualizations. Subspace analysis results are placed in a subdirectory called **SSA**, and the mode visualizations are placed in a subdirectory called **mode_viz**. The exact files that are generated depends on the options that are performed during the analysis.

IV. Additional Types of Analysis

Pooling Data:

It is often useful to pool trajectory statistics. This can be done in JED by combining coordinate files and then performing the usual analysis. To combine the coordinate files, there is a utility program called **Pool_Data.java** that will combine multiple matrices into one. Of courses, the row dimensions of the coordinate files must match (same number of rows). The matrices to combine are specified by an input file called by default **pool.txt** that the user must construct correctly. The program can be run using the following commands:

```
Java -d 64 Pool_Data "/path/to/your/input/file.txt" (This runs the compiled java program)

Java -jar -d 64 Pool_Data.jar "/path/to/your/input/file.txt" (This runs the executable jar file)
```

If you do not pass in the path parameter as an argument at run time, the input file MUST be named pool.txt and be located in the same directory from which the JVM was called.

Below is a sample pool.txt file:

```
-------------------------------------------------------------------------------------------
1
3
/output/directory/
/path/to/first/matrix/              matrix_1.txt
/path/to/first/matrix/              matrix_2.txt
/path/to/first/matrix/              matrix_3.txt
-------------------------------------------------------------------------------------------
```

Notes:
Line #1 specifies the number of jobs (integer) →1
Then for each job you must specify the following:
Line #2 specifies the number of matrices to combine (integer) →3
Line #3 specifies the output directory (string)
Line #4 Token #1 specifies the path to the first matrix (String)
Line #4 Token #2 specifies the name of the first matrix(String)
Line #5 Token #1 specifies the path to the second matrix (String)
Line #5 Token #2 specifies the name of the second matrix(String)
Line #6 Token #1 specifies the path to the third matrix (String)
Line #6 Token #2 specifies the name of the third matrix(String)

Subspace Analysis:

Once JED Driver has been run on multiple trajectories as well as pooled trajectories, an analysis can be done to compare how similar the essential subspaces derived from those trajectories are to each other. JED has a class called **Subspace_Analysis.java** along with 3 driver programs that perform those functions. The core program takes as input two matrices of eigenvectors derived from prior PCA. The matrices must have the same number of rows and columns meaning the vectors being compared come from the same vector space and that the subspaces have the same dimensions. For example, in an analysis of lysozyme you might choose to process 20 cPCA modes while examining 10 different experimental conditions plus pooled data. As long as all the subsets in the analysis are the same then all the 20 dimensional subspaces can be compared.

Like most of the JED programs, the subspace analysis program drivers read an input file called by default **SSA.txt** to obtain runtime information. This file must be constructed properly by the user to perform the analysis. The input file format is the same for all three driver programs.

The 3 driver programs are **SSA_Driver.java**, **FSSA_Driver.java**, and **FSSA_Iterated_Driver.java** and are different only in the kind and type of analysis performed. The SSA_Driver performs a full subspace comparison (non-iterated) including RMSIP, PA, CO, cosine products, and vectorial sum of angles producing both log files and individual flat files. The FSSA_Driver is a light version with only RMSIP and PA output in the log files. The Iterated version performs a recursive variation of the above where all equidimensional subspaces are compared up to the size that was provided, for example, from 1 to 20 by step-size 1 for a 20 column input file.

ALL 3 drivers use the same input file, only the outputs are different.

The driver programs can be run using the following commands:

```
Java -d 64 SSA_Driver “/path/to/your/input/file.txt” (This runs the compiled java program)

Java -jar -d 64 SSA_Driver.jar “/path/to/your/input/file.txt” (This runs the executable jar file)
```

If you do not pass in the path parameter as an argument at run time, the input file MUST be named SSA.txt and be located in the same directory from which the JVM was called.

The format for the SSA.txt file is as follows:

LINE 1: Number_of_Jobs (integer)
LINE 2: Output_Directory (string ending in "/" or "\\")
LINE 3: Batch_Decription (string)
FOR EACH JOB:
Description (string)
Directory1 (string ending in "/" or "\\") Name1 (string)
Directory2 (string ending in "/" or "\\") Name2 (string)

Below is a sample SSA.txt file (for 4 comparisons):

```
--------------------------------------------------------------------------------
4
/output/directory/
Single_Combo_SSA
All-vs-A
/Users/physicslabs/                  all_combo_SS_75_top_20_eigenvectors.txt
/Users/physicslabs/                  1a6n_combo_SS_75_top_20_eigenvectors.txt
All-vs-B
/Users/physicslabs/                  all_combo_SS_75_top_20_eigenvectors.txt
/Users/physicslabs/                  1wit_combo_SS_75_top_20_eigenvectors.txt
All-vs-A+B
/Users/physicslabs/                  all_combo_SS_75_top_20_eigenvectors.txt
/Users/physicslabs/                  1ubq_combo_SS_75_top_20_eigenvectors.txt
All-vs-A_B
/Users/physicslabs/                  all_combo_SS_75_top_20_eigenvectors.txt
/Users/physicslabs/                  1ypi_combo_SS_75_top_20_eigenvectors.txt
--------------------------------------------------------------------------------
```

It is possible to specify hundreds of comparisons when using the driver input file, but beware of the usual gotchas. One typo in a list of one thousand comparisons → CRASH.