SoftSaddle Readme file

Manuel Plasencia Gutiérrez (mpg2@hi.is), Carlos Argáez (carlos@hi.is)

Abstract

SoftSaddle performs saddle point searches in the vicinity of a given minimum. The software uses the Improved Minimum Mode Following method as described in [1]. Bellow is given the basic information needed to execute the code.

Chapter 1

Introduction

This file contains all you need to know to be able to easily run SoftSaddle and compiling its required libraries. To reproduce the results from [1] and [2] please verify the script run.sh according to what it is explained in Sec 3.3. Should you want to run SoftSaddle independently and without script, then please see Chap 5. In this document you will find what libraries you need to run SoftSaddle and how to understand and modify the input file (iniConfig.m), see Chap 4.

Chapter 2

Software requirements

For `SoftSaddle`:

- matlab
  - Parallel Computing Toolbox
  - Statistics Toolbox.

For the libraries to compile:

- gcc
- Blitz++
- Boost
- GSL

For the bashscript `run.sh` to run your shell must recognise the next bash commands (basic commands in bash, commonly accessible but just making sure):

- lscpu
- sed
- if, then, else, fi commands
- basic bash multiplication
- ldconfig
- cut
- grep
- exit

- echo

Make sure that the command:

```
$ matlab
```

works properly in your computer. You can check that by doing:

```
$ which matlab
```

and it should print the address where the `matlab` executable is located. Otherwise, keep in mind where the `matlab` executable is located for later you will need to edit the `run.sh` script. That is something very simple to do: See Sec 3.3.

Likewise, make sure that the command:

```
$ g++
```

works properly in your computer. You can check that by doing:

```
$ which g++
```

Make sure that blitz++, boost, GSL libraries are properly exported in your bash so that they are accessible in your system. Otherwise, export them in `run.sh`, see Sec 3.3.

Chapter 3

About the `run.sh` script

3.1 Before running the script `run.sh`

`run.sh` is a bashscript written to automatically run `SoftSaddle.tar.gz` to reproduce the results in [1] and [2]. Notice that it also untars the file `SoftSaddle.tar.gz` before it starts running the problems in [1] and [2]. So it is written to do EVERYTHING. However, there are few requirements for it to work. All those requirements will be clearly explain in this document.

DO NOT USE FOLDER NAMES WITH SPACES LIKE ”this folder”. Instead make sure that all your folders' names have NO spaces like ”thisfolder”. IMPORTANT!!

In `SoftSaddle` one can control the number of searches of Saddle Points by controlling the total amount of processors you will use and the number of attempts you want to perform. For example, if you will use only 4 processors and you want to perform 100 total Saddle Points searches, then you need to do 25 total attempts because:

$$4 \times 25 = 100 \text{ searches}$$

Then the 100 searches are distributed in all 4 processors by an equal amount of attempts.

In the particular case of [1], the total number of processors used was 25 and the total attempts was 220, which makes:

$$25 \times 220 = 5500 \text{ searches}$$

INFORMATION EXTRA 3.1.1 *There is an extra number you need to be acquaintance of because* `run.sh` *will use it: "maximum number of processors this computer has". That is just the amount of processors the computer has and it should be greater or equal to the number of processors you will use, obviously.*

$$\textit{Number of processors available in the computer} \geq \textit{Number of processors to used}$$

`run.sh` utilizes `lscpu` to find out how many “cores per socket” your computer has and accordingly it sets the input files.

Furthermore, it does set all folders, compiles the libraries (**provided** your computer has all libraries properly installed and exported), run the problems and gives you the results inside a folder named RESULTS.

3.2 Understanding the output of the script `run.sh` for your system

The script `run.sh` will do several things, they are enumerated here:

1. It untars the file `SoftSaddle.tar.gz` where are located the code, the libraries and the problems to be run
2. It makes the directory ”`RESULTS`” inside the main-untared directory ”`SoftSaddle`”
3. it sets the INPUT files to be executed
4. It goes to the corresponding folders and compile the first library (the library containing the mixture of Davidson and Lanczos algorithms) and places the compiled library in the corresponding folder
5. It compiles the Lanczos library and places it on the right folder
6. It computes the problem ”`adaptive_HSphere-lbfgs-davidson-sm50`”
7. It executes the matlab script: ”`get_data_for_web_benchmark.m`”, which analyses the results for `adaptive_HSphere-lbfgs-davidson-sm50`
8. It computes the problem ”`adaptive_HSphere-lbfgs-lanczos-sm0`”
9. It executes the matlab script: ”`get_data_for_web_benchmark.m`”, which analyses the results for `adaptive_HSphere-lbfgs-lanczos-sm0`
10. It computes the problem ”`adaptive_HSphere-lbfgs-lanczos-sm50`”
11. It executes the matlab script: ”`get_data_for_web_benchmark.m`”, which analyses the results for `adaptive_HSphere-lbfgs-lanczos-sm50`
12. It finishes and exits
13. The results will be on the folders inside `SoftSaddle/RESULTS` and they are saved in the file with extension `*.con`

3.3 Setting the script `run.sh` for your system

This bash scripts can only work if the libraries

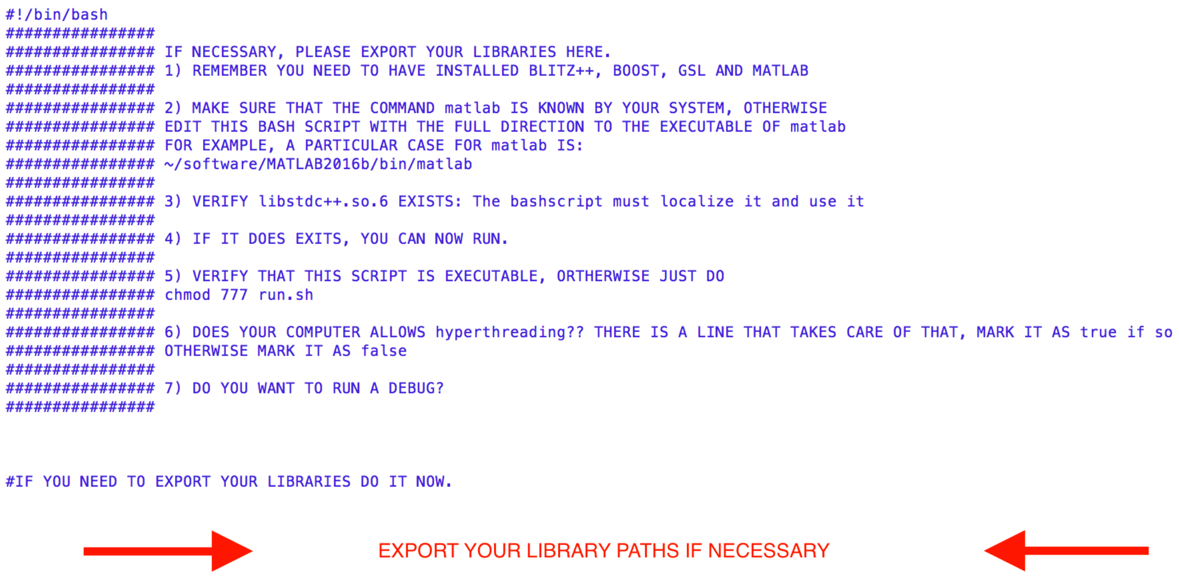
- gcc
- Blitz++
- Boost
- GSL

are properly installed and exported in your computer. Make sure that the bash commands

- lscpu
- sed
- if
- basic bash multiplication

are also available in your shell. Now, follow the next instructions:

Should you need to export the libraries gcc needs to compile, then please do it in the bash script, do it in the same section shown in figure 3.4.



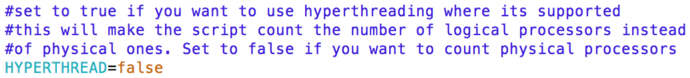
```
#!/bin/bash
################
################ IF NECESSARY, PLEASE EXPORT YOUR LIBRARIES HERE.
################ 1) REMEMBER YOU NEED TO HAVE INSTALLED BLITZ++, BOOST, GSL AND MATLAB
################
################ 2) MAKE SURE THAT THE COMMAND matlab IS KNOWN BY YOUR SYSTEM, OTHERWISE
################ EDIT THIS BASH SCRIPT WITH THE FULL DIRECTION TO THE EXECUTABLE OF matlab
################ FOR EXAMPLE, A PARTICULAR CASE FOR matlab IS:
################ ~/software/MATLAB2016b/bin/matlab
################
################ 3) VERIFY libstdc++.so.6 EXISTS: The bashscript must localize it and use it
################
################ 4) IF IT DOES EXITS, YOU CAN NOW RUN.
################
################ 5) VERIFY THAT THIS SCRIPT IS EXECUTABLE, ORTHERWISE JUST DO
################ chmod 777 run.sh
################
################ 6) DOES YOUR COMPUTER ALLOWS hyperthreading?? THERE IS A LINE THAT TAKES CARE OF THAT, MARK IT AS true if so
################ OTHERWISE MARK IT AS false
################
################ 7) DO YOU WANT TO RUN A DEBUG?
################


#IF YOU NEED TO EXPORT YOUR LIBRARIES DO IT NOW.
```

EXPORT YOUR LIBRARY PATHS IF NECESSARY

Figure 3.1: Section in which you would export your library paths, if necessary.

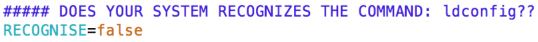
Does your computer allows hyperthreading?



```
#set to true if you want to use hyperthreading where its supported
#this will make the script count the number of logical processors instead
#of physical ones. Set to false if you want to count physical processors
HYPERTHREAD=false
```

Figure 3.2: If your computer allows hyperthreading, then modify this into true.

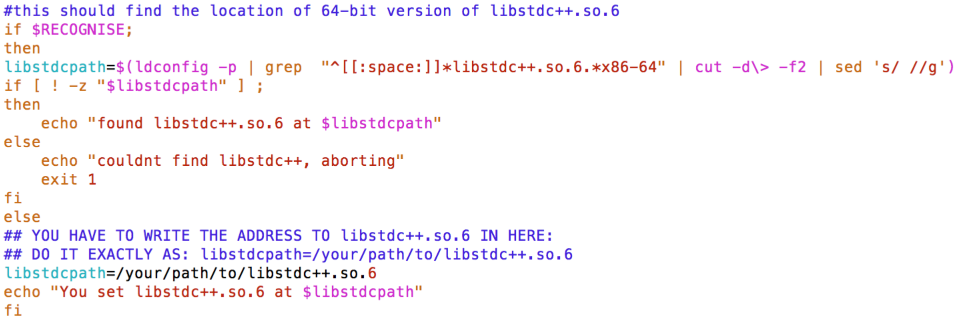
Does the running computer allows the command `ldconfig -p`?



```
##### DOES YOUR SYSTEM RECOGNIZES THE COMMAND: ldconfig??
RECOGNISE=false
```

Figure 3.3: If the running computer allows the command `ldconfig -p` set this to true.

Otherwise, if it does not allows the command `ldconfig -p`, then you need to set it as false and then use the whole address to the library `libstdc++.so.6` as:



```
#this should find the location of 64-bit version of libstdc++.so.6
if $RECOGNISE;
then
libstdcpath=$(ldconfig -p | grep  "^[[:space:]]*libstdc++.so.6.*x86-64" | cut -d\> -f2 | sed 's/ //g')
if [ ! -z "$libstdcpath" ] ;
then
    echo "found libstdc++.so.6 at $libstdcpath"
else
    echo "couldnt find libstdc++, aborting"
    exit 1
fi
else
## YOU HAVE TO WRITE THE ADDRESS TO libstdc++.so.6 IN HERE:
## DO IT EXACTLY AS: libstdcpath=/your/path/to/libstdc++.so.6
libstdcpath=/your/path/to/libstdc++.so.6
echo "You set libstdc++.so.6 at $libstdcpath"
fi
```

Figure 3.4: If the running computer does not allows the command `ldconfig -p`, then you need to set it as false and then use the whole address to the library `libstdc++.so.6`.

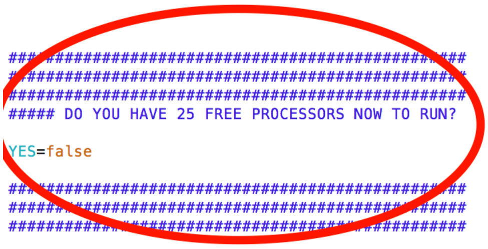
The bash script automatically checks for how many “cores per socket” or for how many “hyperthreading” cores your computer has. If you allow `HYPERTHREAD=true` then it will check for hyperthreading, if `HYPERTHREAD=false`, it will use physical cores. Accordingly it will automatically set all parameters to run the results in [1] and [2] and it will find:

- The number of processors the computer will use
- The number of attempts accordingly to reproduce the results in [1] and [2]
- The number of processors the computer has

Using the bash command `lscpu`, our script `run.sh` will automatically decide how many processors it will use and it will automatically set the input file `iniConfig.m` accordingly. Should you want to modify manually `iniConfig.m` , see Sec 3.3. Notice that in [1] and [2], the results were obtained by running 5500 searches distributed in 25 processors with 220 attempts.

NOTICE THAT: `run.sh` WILL NOT VERIFY IF YOU HAVE 25 AVAILABLE PROCESSORS TO RUN. So, if you want to use 25 processor to identically compute the results as in [1] and [2], then read 3.3.1:

IMPORTANT 3.3.1 *The bash script has a line that asks whether you have 25 available processors in your machine to run all our computations. Should you have those available 25, please edit the bash script with* ***`vim`*** *or any other editor you would prefer, and change the line YES=`false` to YES=`true`. See the next figure:*



```
#################################################
#################################################
#################################################
##### DO YOU HAVE 25 FREE PROCESSORS NOW TO RUN?

YES=false

#################################################
#################################################
#################################################
```

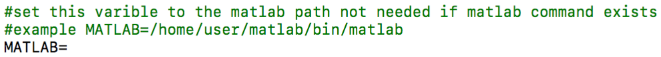
Figure 3.5: If you have 25 available processors to run, change `false` to `true` as in the figure.

In case your computer does NOT have 25 free processors to run the results in [1] and [2], then do NOT modify the ***`run.sh`*** *for it will set things automatically.*

The bashscript will verify where is located the library `libstdc++.so.6`. Commonly, it is located in `/usr/lib64/libstdc++.so.6` but according to the system libraries it could also be located in folders like: `/usr/lib/x86_64-linux-gnu/libstdc++.so.6`. So, the bash script will look for it and it will settle it for running with the actual directory address of your system: `/your/system/directory/libstdc++.so.6`. It is a good practices though to verify where is installed in case you would need to edit the bashscript manually.

Now, `run.sh` will call `matlab` to execute `SoftSaddle`. If the command `$ matlab` works in you shell without any other requirement, then this part is set. However, should you need to use the whole path to be able to call the executable of `matlab`, as `/home/user/matlab/bin/matlab` then do the following:

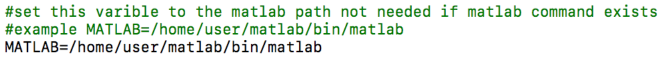
Localize the section in the bashscript where you must define `matlab` whole path, see figure 3.6.



```
#set this varible to the matlab path not needed if matlab command exists
#example MATLAB=/home/user/matlab/bin/matlab
MATLAB=
```

Figure 3.6: Example of how looks the call to `matlab`.

And replace "`matlab`" with the whole address to the `matlab` executable `/the/whole/path/to/call/matlab`. See figure 3.7 in which one particular example is given.



```
#set this varible to the matlab path not needed if matlab command exists
#example MATLAB=/home/user/matlab/bin/matlab
MATLAB=/home/user/matlab/bin/matlab
```

Figure 3.7: Particular example, very pedagogical, of how to modify the script to call the executable of `matlab`.

In a similar way as explained before, verify the proper location of the command:

```
g++
```

Should you change it, just do it in the equivalent way you have already done it for `matlab`.

3.4 Run the script `run.sh`

This script will run `SoftSaddle` and compile all required libraries. According to the script, the libraries will be placed in the right folders and the correct paths will be given to all files that required them.

To run it, just do:

```
$ ./run.sh
```

in the same folder where is located `SoftSaddle.tar.gz`.

If the bashscript is not executable, then just do:

```
$ chmod 777 run.sh
```

That will ensure that the script is executable, so it can be run.

3.5 Run a debug text with `run.sh`

Before you do any computation, check that everything looks OK. Edit `run.sh` and look for the line:

```
RUNDEBUG=false
```

and change it for

```
RUNDEBUG=true
```

this will check that all libraries are properly found and it will tell you how many processors it will use.

IMPORTANT 3.5.1 *IF THE COMMAND `matlab` IS NOT FOUND THE BASH-SCRIPT WILL EXIT REGARDLESS OF YOU HAVE CHOSEN TO DEBUG OR NOT. SO MAKE YOUR THE `matlab` COMMAND IS PROPERLY SET IN YOUR COMPUTER.*

3.6 Run a test with `run.sh`

It will be good to run a small test with this bashscript. This is a small computation that should last few minutes (it is been measured to be less than 5 minutes) and it will run the bashscript, compile the libraries, set the input file, set the `SoftSaddle.m` to run, and execute the script to analyse the results. If it runs fine the it means you are set to reproduce the long computation results (those take much longer). To do this checking, just go to the line `RUNFASTTEST=false` and change it by `RUNFASTTEST=true`

Then the bash script will automatically set the input files to run 1 max attempt with 2 processors (2 saddle point searches it total) with 10 repetitions.

If you don't want to run this test, just make sure that this option is set as false: `RUNFASTTEST=false`

Once you have completely run the test with satisfaction, then just remove the folder `SoftSaddle`, set the option to `false` and run, be patient because it will take long time.

Chapter 4

About the code `SoftSaddle`

Although the bash script `run.sh` takes care of everything. It is usually a very good practise to know what is what we are running and how to modify the required parameters. However, for first use `run.sh` takes cares of everything.

4.1 SoftSaddle

`SoftSaddle` is a code written to find Saddle Points on potential energy surfaces (PES). It needs to read two c++-compiled libraries `liblanczos.so` or `libdavidson.so` and `libmorse.so`. These are shared libraries which are written using the libraries Blitz++, Boost and GSL. `SoftSaddle` requires to be set in a way that `matlab` knows where to read the `SoftSaddle` files when `SoftSaddle.m` is called. Obviously, `SoftSaddle.m` is the matlab-function to be called when executing `SoftSaddle`.

`SoftSaddle` requires to read the input file called: `iniConfig.m`. All parameters when running `SoftSaddle` must be set in `iniConfig.m`. However, the files of `SoftSaddle` must be placed in a general folder with known direction while `iniConfig.m` is placed in the folder where we want the results to be computed. Therefore, it is recommended to use `startup.m` in which the address to the files of `SoftSaddle` is clearly set.

The contain of `startup.m` is only the next line:

```
addpath('/the/address/where/SoftSaddle/is/');
```

The `iniConfig.m` is the input file. All settings must be clear there and it can be divided in 6 sections. In the following, the variables of each section will be explained.

4.1.1 Simulation

In this section, we have to put the values for different variables that refer to the simulation itself.

NOTE: EVERY LINE MUST END WITH A ; (SEMICOLON)

- desired_number_sp =
  This sets the total number of wanted saddle points. When that amount of saddle points is obtained, the simulation stops. Preset to: 10500;

- parall_search_per_min=
  This is the number of parallel searches that SoftSaddle will carry out. Each one of them will use one processor, so it could be said as well to be the number of processors we will use. Preset to: 25;

- max_attempts =
  This is the total number of attempts SoftSaddle will carry out. It will search for saddle points up to this number per each repetition. Preset to: 220;

- real_n_cores=
  This is the total number of processors the computer has. Preset to: 25;

- RUNTIME =
  This is the total time in seconds that the computation will run. When this seconds are finished, the computation stops. Preset to: 86400*10;

- absTol =
  This is the tolerance in distances between two points to be consider the same point. Less than or equal to mean: "Same point" while bigger that this mean "Different points". Preset to: 0.1;

- delta =
  This is the MAXIMUM displacement allowed per iteration, measured in Amstrongs. It means that this is the MAXIMUM allowed distance between two points when computing the minimum mode. Preset to: 0.5;

- sigma =
  This is the standard deviation for gaussian. Preset to: 0.3;

- a_to_disp =
  This is the number of atoms (each atom has three coordinates) in the island to be displaced. Preset: 7;

- pick_rand_atoms =
  This allows to displace random atoms: Preset false; because we want to displace those on the top.

- displace_radius =
  This is the variable that controls the distance from atoms in the centre. All between this distance from the centre will be include in the displacement.

- get_ini_guess =
  This means that SoftSaddle HAS to read the configuration from the given file in the SoftSaddle folder. Preset to: true;

- bench_energy =
  This is the number for the energy, which means that SoftSaddle will write a report (on screen) with the number of atoms with this energy. Preset to: 4.0;

- ref_energy =
  This energy will be used as stopping criterium for the web benchmark. Preset to: 1.5;

- distribution =
  Variable to specify type of distribution for random displacements
  Preset to: 1;
  distribution=1 : Gausian distribution
  distribution=2 : Uniform dist in n-1 subspace (number will be uniformly distributed arround an empty hypersphere with radius sigma)
  distribution=3 : Uniform (a number between -1.0 and 1.0 is generated from uniform distribution, then multiply by sigma)
  distribution=4 : Similar to 2 but relative to each displaced atom rather to relative to the system. Uniform distribution in n-1 subspace (number will be uniformly distributed around an empty hypersphere with radius sigma from each atom)

- f_name = 'morse ';
  This is the potential energy name. This is a name you will give to identify your problem: User defined.

- write_out_to_hdd =
  This is a boolean variable to define if the detailed results should be written in the harddisk. If chosen as true, they will be saved inside the folder where the iniConfig.m is. Preset to: false;

- new_displacement_per_tune =
  This is a boolean variable. It sets if each repetition uses different starting points for SP-searches. Preset to: true;

TO BE NOTICED:
The number of searches of Saddle Points depends on the total amount of processors you will use and the number of attempts you want to perform. For example, if you will use only 4 processors and you want to perform 100 total Saddle Points searches, then you need to do 25 total attempts because:

$$4 \times 25 = 100 \text{ searches}$$

Then the 100 searches are distributed in all 4 processors by an equal amount of attempts; in the particular case of [1], the total number of processors used was 25 and the total attempts was 220, which makes:

$$25 \times 220 = 5500 \text{ searches}$$

4.1.2 Hyper sphere

These are the variables that control the hyper sphere.

- disp_by_hypersphere_method =
  This is a boolean variable to control if the initial displacements are generated on the hyperspherical surface or not. Preset to: true;

- hyper_rad0 =
  This variable sets the initial radio of the hyper sphere. Preset to: 0.1;

- read_from_database =
  This is a boolean variable to control if the distributed points on the hypersherical surface are going to be recomputed for new computations, or if they will be read from the preloaded data base. Preset to: true; The bash script run.sh takes cares of this: If the number of displacements is unknown in the data base, the bashscript changes to false, otherwise it changes to true.

- self_learning_hyper_rad =
  Boolean variable if chosen true: it uses variable hypersphere radius. If chosen false, it uses constant hypersphere radius: R = hyper_rad0.

- automatic_self_learning =
  Preset to: true;False means a predefined scheduled for varying the hypersphere R used. True means that the distances to sp and λ_0 surfaces are used to vary the hypersphere R.

- self_learning_scheme = 2;
  Different ways to estimate change in hyper R. To reproduce the results in [1] and [2] use: 2.

  1: use minimum distance to lambda zero
  2: use average distance to lambda zero
  3: use maxima distance to lambda zero
  4: use average of minima distances
  5: use average of average distances
  6: use average of maxima distances

- mix_prev_hyper_rad = true;
  True: average previous hyper R in the schemes (smoother transition from one hyper R to next one)

- sort_displacements = true;
  True means displacements are sorted by furthest distances and chosen accordingly False means the points are chosen randomly.

4.1.3 Minimizer

These are the variables that control the hyper sphere.

- min_method =
  This variable sets one of the two possible methods: LBFGS or CG (Conjugate Gradient). Preset: 'LBFGS';
- memory_size =
  This variable sets the size of matrix that we want to preserved in memory for the case of LBFGS. Preset: 60;
- finite_diff_step =
  This is the delta for the finite difference method. Preset: 1e-4;
- cg_iter_max =
  This is the maximum number of iterations for CG. Preset: 1000;
- ls_iter_max =
  This is the maximum number of line search iterations. Preset: 1;
- cg_err =
  This is the CG error tolerance. Preset: 1e-3;
- ls_err =
  Line search error tolerance. Preset: 1e-2;
- max_step_size =
  This variable is defined in SoftSaddle and it must be set to delta. Preset: delta ;
- RESET =
  Boolean variable: if true CG is reset when lambda zero boundary is crossed. Preset: false;
- get_min_mode_ =
  Boolean variable. Set true to get the minimum mode. Preset: true;
- normalize_direction =
  Boolean variable: if true the search direction is normalized. Preset: true;
- maximum_speed =
  Preset: false;
- dE_max =
  If dE goes beyond this value the search is aborted. Preset: 480.0;
- obligate_move =
  Boolean variable. Preset: true;
  True means the SP-search will be forced to initiate when the initial configuration

is very close to the minimum and where the gradient is smaller that the stopping tolerance.

- use_same_formula =
  Boolean variable. Preset: false;
  True means using same MMF formula when first exit positive area
- alf =
  Preset: [];
  if equal zero -gradient is used as direction to escape positive area
- n_intentos =
  Preset: 0;
  Attempts to force the system staying inside SP basin of attraction.
- avoid_stuck_in_min =
  Bolean variable: Preset: true;
  True means convergence in area with positive eigenvalue will be rejected and the system forced to keep searching for SPs. False means convergence in area with positive eigenvalue will be rejected and the search will stop with a non-converged message.

4.1.4 Eigenvectors: Lanczos, Davidson, etc

These are the variables that control Lanczos or Davidson algorithm.

- lib_name=
  Name of the *.so library to be read. Preset to: 'libdavidson'; It can also be: 'liblanczos';
- lib_path=
  Direction where the library will be read.
- lanc_iter =
  Maximum iterations to compute minimum mode using Lanczos or Davidson. Preset to: 8;
- lanc_abs_tol =
  Covergence tolerance for eigenvalue. Preset to: 0.01;
- lanc_finite =
  Variable to control the finite difference step. Preset to: finite_diff_step;
- save_lanczos_call =
  Boolean variable: if true, the minimum mode is not calculated when the step is smaller than a given percent of maximum step size. Preset to: true;
- save_lanczos_factor =
  Percent of maximum step size (recomended 50% for max_step_size=0.5) 50.0;

4.1.5 Conjugate Grad

These are the variables that control the conjugate gradient.

- beta_formula=9;
  1: Hestenes and Stiefel
  2: Polak-Ribiere
  3: Liu and Storey
  4: Dai-Yuan
  5: Fletcher-Reeves
  6: conjugate descent
  7: Hestenes-Stiefel + Dai-Yuan
  8: Polak-Ribiere + Fletcher-Reeves
  9: Liu-Storey + conjugate descent

- modified_CG =
  Boolean variable. Preset to: false;
  Uses a modified formula to compute search direction.

- e_tol =
  Tolerance to evaluate uniform descent condition in CG. Preset1e-8;

- get_min_mode_lineSearch =
  This variable is boolean. If true then calculates the minimum mode if needed. Preset to true;

4.1.6 Repetitions for statistical analysis

These are the variables that control the repetitions.

- do_repetition =
  Boolean variable. Preset true;
  This makes repetitions happen. When false, it does not make any repetition.
  n_repetition =
  This the the number of repetitions that must be carry on. Preset to: 10;

Chapter 5

Generalities about `SoftSaddle`

5.1 Setting up the simulation variables

Edit the file iniConfig.m as extensively explained chapter 4

5.2 Execute the code

Should you execute the code manually rather than using the run.sh script, then follow the next small algorithm to proceed. NOTE THAT TWO FILES MUST BE CONTAINED IN THE FOLDER YOU ARE ABOUT TO RUN: `iniConfig.m` AND `startup.m`. The first one is the input file, the second one is the file where `matlab` finds out where to read `SoftSaddle`.

1. Place the file `SoftSaddle.tar.gz` in the folder you want and untar it: `tar xvfz SoftSaddle.tar.gz`
2. `cd SoftSaddle && mkdir RESULTS` this enters the untarred `SoftSaddle.tar.gz` file and creates the folder `RESULTS`
3. To compile `libdavidson.so` (which is the library to run Lanczos/Davidson algorithm for computing the lowest eigenvalue) move to the folder where the libraries are:

   ```
   cd libraries/davidsson_lib/to_compile/
   ```

   and compile the `libdavidson.so`. This is a shared library which is compiled with the next command:

   ```
   g++ -O3 -shared -Wl,-soname,libdavidson.so -o libdavidson.so -fPIC *.cpp
   tridiag.c tridiagv.c -lgsl -lgslcblas
   ```

   Make sure that all libraries are properly exported: `Blitz++, Boost, GSL.` Should this compilation fail, then check your library path.

4. copy `libdavidson.so` to the folder where the `source_code` is. From your last location it should be:

   ```
   cp libdavidson.so ../../../source_code/.
   ```

5. Move to the `lanczos_lib` folder and compile the `liblanczos.so` library:

   ```
   cd ../../lanczos_lib/to_compile/
   ```

   ```
   g++ -O3 -shared -Wl,-soname,liblanczos.so -o liblanczos.so -fPIC *.cpp
   tridiag.c tridiagv.c -lgsl -lgslcblas
   ```

6. Copy `liblanczos.so` to the folder where the `source_code` is. From your last location it should be:

   ```
   cp liblanczos.so ../../../source_code/.
   ```

7. Move to the `libmorse` folder. From your last location, it should be: `cd ../../libmorse/`

8. Compile the libmorse.so library with the next command:

   ```
   g++ -O3 -shared -Wl,-soname,libmorse.so -o libmorse.so -fPIC *.cpp -lgsl
   -lgslcblas
   ```

9. Copy `libmorse.so` to the folder where the source_code is. From your last location it should be:

   ```
   cp libmorse.so ../../source_code/.
   ```

10. Now you need go back to SoftSaddle untarred folder. From your last location it should be:

    ```
    cd ../../.
    ```

11. Copy the folders in test_run/paper-result-reproduce/ into RESULTS/. To do that, you can use the next instruction:

    ```
    cp -r test_run/paper-result-reproduce/* RESULTS/
    ```

    Enter into `RESULTS`, do:

    ```
    cd RESULTS
    ```

12. To Run `adaptive_HSphere-lbfgs-davidson-sm50`:
    Enter to the folder:
    `adaptive_HSphere-lbfgs-davidson-sm50`: `cd adaptive_HSphere-lbfgs-davidson-sm50`
    Now, generate the startup.m file. Do as follows:

```
cd ../../source_code/
address=$(pwd)
cd -
echo "addpath('"$address"');" > startup.m
```

This generates a file named `startup.m` whose solemnly content is the line `addpath('/the/path/to/source_code');`

You can also do it manually, just go the folder where the `source_code` is and use the command `pwd`. Then just copy that address into your `startup.m` manually.

13. Modify the `iniConfig.m` in the line `lib_path=` and write the address where the `source_code` is. Now you can simply do it as:

    ```
    address2=$(echo "lib_path='"$address"';")
    sed -i "s|lib_path=.*|$address2|g" iniConfig.m
    ```

    You can also do it manually, just go the folder where the `source_code` is and use the command pwd. Then just copy that address into your iniConfig in the line `lib_path='/the/path/to/source_code'` manually.

14. Make sure that the line `lib_path` is set as follows: `lib_name='libdavidson';`. You can do it manually using any text editor your prefer or by doing:

    ```
    sed -i "s/lib_name=.*/lib_name='libdavidson';/g" iniConfig.m
    ```

    NOTICE THE UNDERLINE "_".

15. Again in `iniConfig.m`, you need to modify 3 parameters.
    We want 5500 displacements to be able to reproduce the results in [1] and [2]. So, how many processor does the computer have? then change the variable

    `parall_search_per_min=` with the number of processors you will use `max_attempts=` with the maximum attemps you will allow. Remember that $parall\_search\_per\_min \times max\_attempts = 5500$ This means that if you have 8 processors you can set

    ```
    parall_search_per_min=5
    max_attempts=1100
    real_n_cores=8
    ```

    Or you could also set

    ```
    parall_search_per_min=4
    max_attempts=1375
    real_n_cores=8
    ```

    Now, you can just run.

    ```
    LD_PRELOAD="/usr/lib64/libstdc++.so.6" matlab -r "SoftSaddle"
    ```

    This will run the program. NOTICE THE UNDERLINE "_".
    If `libstdc++.so.6` is installed in other directory then you need to run:

    ```
    LD_PRELOAD="/your/system/directory/libstdc++.so.6" matlab -r "SoftSaddle"
    ```

16. To Run `adaptive_HSphere-lbfgs-lanczos-sm0` or `adaptive_HSphere-lbfgs-lanczos-sm50`: Just do exactly the as for `adaptive_HSphere-lbfgs-davidson-sm50` but be sure that in the iniConfig.m the line `lib_name` is set as follows: `lib_name='liblanczos';`

5.3 Other generalities

- Always execute the code from within the folder `RESULTS`.
- To execute the code interactively open a `matlab` session and type:

```
[ lowestE, LowestX, Elist, Xlist, Esplist, Xsplist, Tableconnected ] = SoftSaddle;
```

- To execute the code in the background or to submit it to a queue system: `matlab -r "SoftSaddle;"`
- Results are saved inside the folder RESULTS. A folder named `0_0_0` is created with the results of a first local minimization preformed to bring the system to a local minimum. Then a new folder is created for each saddle search conducted to save the relevant details.
- A summary of the simulation is also saved in RESULTS, for instance a table showing the relevant information about each saddle point search (i.e., computational cost, energy, wether the saddle is connected or not, etc).
- Source code: The code is mainly written in Matlab. However the computation of the lowest eigenvector of the Hessian matrix is conducted in an external C++ library. The path to the library may be defined in the iniConfig.m file, otherwise the library should be in the same directory with all source code.
- The Davidson method for computing the lowest eigenvector is implemented as described in [2]. This library can be found under the name file name `libdavidson.so`

5.4 The Pt heptamer island on a Pt(111) surface

The initial configuration of this system is read from hard drive at the beginning of the simulation: Pt island iniConfiguration/displace bech.mat The file displace bech.mat is formated in a way that has six columns and many rows. Each row represents one atom. Column 1, column 2 and column 3 are x, y and z component respectively. Column 5 indicates wether the atom is free to move or not. 0 means free to move and 1 means frozen. Column six represents atom ID and it goes from 0 to N-1 where N is total number of atoms. The file can be edited in `matlab`.

Chapter 6

Possible failure causes

6.1 Libraries

Did you install Blitz++, Boos, GSL?

If so, Are they properly exported so that the system knows where to find them?

6.2 Libraries during compilation

```
davidson.cpp:24:25: fatal error: blitz/array.h: No such file or directory
#include <blitz/array.h>
                          ^
compilation terminated.
matlab.cpp:1:25: fatal error: blitz/array.h: No such file or directory
#include <blitz/array.h>
                          ^
compilation terminated.
In file included from tools.cpp:20:0:
tools.hpp:4:30: fatal error: boost/function.hpp: No such file or directory
#include <boost/function.hpp>
                              ^
compilation terminated.
tridiag.c:22:26: fatal error: gsl/gsl_math.h: No such file or directory
#include <gsl/gsl_math.h>
                          ^
compilation terminated.
tridiagv.c:22:26: fatal error: gsl/gsl_math.h: No such file or directory
#include <gsl/gsl_math.h>
                          ^
compilation terminated.
cp: cannot stat ‘libdavidson.so’: No such file or directory
```

This error occurs for two possible reasons:

1. you don’t have properly install all libraries
2. you did install properly all libraries but you fail to export them

6.3 Error while loading shared libraries: libstdc++.so.5

Please verify that you a using matlab with your system’s `libstdc++.so.5` library. The bash script `run.sh` should be checking where you have such library with the command:

```
ldconfig -p | grep  "^[[:space:]]*libstdc++.so.6.*x86-64" | cut -d\> -f2 | sed ’s/ //g’
```

and it should using `matlab` with such library with the command

6.4 bash: matlab: command not found

Make sure you have set properly the `matlab` command in the script. At the beginning of the script there is a section:

```
#set this varible to the matlab path not needed if matlab command exists
#example MATLAB=/home/user/matlab/bin/matlab
MATLAB=
```

In there you need to add the whole address to your matlab

```
#set this varible to the matlab path not needed if matlab command exists
#example MATLAB=/home/user/matlab/bin/matlab
MATLAB=/whole/address/to/your/matlab
```

If the command `$which matlab` works in your computer, then this should not be the problem

6.5 Error in SoftSaddle (line 422) mypool=parpool(real_n_cores);

This error comes because your computer’s logical processors don’t allow multithreading with hyperthreading. To avoid this, use physical processors instead. So, at the beginning of the script, go to the section:

```
#set to true if you want to use hyperthreading where its supported
#this will make the script count the number of logical processors instead
#of physical ones. Set to false if you want to count physical processors
HYPERTHREAD=true
```

and change `HYPERTHREAD=true` to `HYPERTHREAD=false`

Bibliography

[1] Manuel Plasencia Gutiérrez, Carlos Argáes and Hannes Jónsson *Improved Minimum Mode Following Method for Finding First Order Saddle Points*, Journal of Chemical Theory and Computation Article ASAP, 2016, DOI: 10.1021/acs.jctc.5b01216

[2] Carlos Argáes, Manuel Plasencia Gutiérrez and Hannes Jónsson *Improved Minimizer Methodology for Finding Low Energy First Order Saddle Points*, In preparation.