SMG2S Manual

For SMG2S Release 1.0.0

Version 1.0

Xinzhe Wu

August 24, 2018

Abstract

Iterative linear algebra methods are the important parts of the overall computing time of applications in various fields since decades. Recent research related to social networking, big data, machine learning and artificial intelligence has increased the necessity for non-hermitian solvers associated with much larger sparse matrices and graphs. The analysis of the iterative method behaviors for such problems is complex, and it is necessary to evaluate their convergence to solve extremely large non-Hermitian eigenvalue and linear problems on parallel and/or distributed machines. This convergence depends on the properties of spectra. Then, it is necessary to generate large matrices with known spectra to benchmark the methods. These matrices should be non-Hermitian and non-trivial, with very high dimension. A scalable parallel matrix generator SMG2S that uses the user-defined spectrum to construct large-scale sparse matrices and ensures their eigenvalues as the given ones with high accuracy is implemented based on MPI and C++11. This report gives the manual of SMG2S.

Contents

Chapter 1

Introduction

1.1 Getting Started

SMG2S (Scalable Matrix Generator with Given Spectrum) is a software which provides to generator the non-Hermitian Matrices with User-customized eigenvalues. SMG2S is implemented in parallel based on MPI (Message Passing Interface) and C++11 to support efficiently the generation of test matrices on distributed memory platforms.

Iterative linear algebra methods are important for the applications in various fields. The analysis of the iterative method behaviors is complex, and it is necessary to evaluate their convergence to solve extremely large non-Hermitian eigenvalue and linear problems on parallel and/or distributed machines. This convergence depends on the properties of spectra. Thus, we propse SMG2S to generate large matrices with known spectra to benchmark these methods. The generated matrices are non-Hermitian and non-trivial, with very high dimension. SMG2S can generate the non-Hermitian matrices with user-custormized eigenvalues.

The ability of SMG2S to keep the accuracy of given spectrum can be verified by the functionality proposed inside SMG2S. This function is based the shift inverse power method. SMG2S gives also an graphic user interface to compare the given and final spectral distribution for the verification.

We will describe the following subset of the SMG2S.

- **Parallel Vector and Matrix:**
- **Nilpotency Matrix Object:**
- **Generating Matrix with predescribed eigenvalues:**
- **Interface to Other Languages/Libraries:**
- **Verification of Eigenvalues of Generated Matrix:**

1.2 Installation

To obtain SMG2S, please follow the instructions at the SMG2S download page: https://smg2s.github.io/download.html.

Prerequisites:

- C++ Compiler with **c++11** support;
- Cmake (version minimum 3.6);
- (Optional) PETSc and SLPEc are necessary for the verification of the ability to keep the given spectrum.

Int the main directory:

```
cmake . -DCMAKE_INSTALL_PREFIX=${INSTALL_DIRECTORY}
```

The main.cpp will generate an exectutable smg2s.exe to demonstrate a minimum sample :

```
make
```

The main part of SMG2S is a colllection of header files. Install the header files into ${INSTALL_DIRECTORY}

```
make install
```

For testing:

```
make test
```

The output of test should be like:

```
Running tests...
Test project /User/name/SMG2S
Start 1: Test_Size_10000_w_proc1
1/4 Test #1: Test_Size_10000_w_proc1 .. Passed 1.20 sec
Start 2: Test_Size_20000_w_proc2
2/4 Test #2: Test_Size_20000_w_proc2 .. Passed 1.22 sec
Start 3: Test_Size_10000_s_proc1
3/4 Test #3: Test_Size_10000_s_proc1 .. Passed 1.20 sec
Start 4: Test_Size_10000_s_proc2
4/4 Test #4: Test_Size_10000_s_proc2 .. Passed 0.66 sec

100% tests passed, 0 tests failed out of 4

Total Test time (real) =   4.29 sec
```

1.3 CMake Options

We uses CMake is to build, test and package SMG2S. If you do not have PETSc and SLEPc in your platform, please make sure the option below is OFF.

```
option(INSTALL_TO_USE "Install SMG2S include files?" OFF)
```

1.4 Copyright and Licensing of SMG2S

SMG2S is a open source software published under the GNU Lesser General Public License v3.0. SMG2S can be redistributed and modified under the terms of this license.

SMG2S is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. SMG2S is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with SMG2S. If not, see ¡http://www.gnu.org/licenses/¿.

1.5 Programming Language in SMG2S

SMG2S is a collection of templated header files written in C++. The wrappers to C and Python codes are provided. The users of PETSc or Trilinos can directly use SMG2S with the interfaces implemented inside.

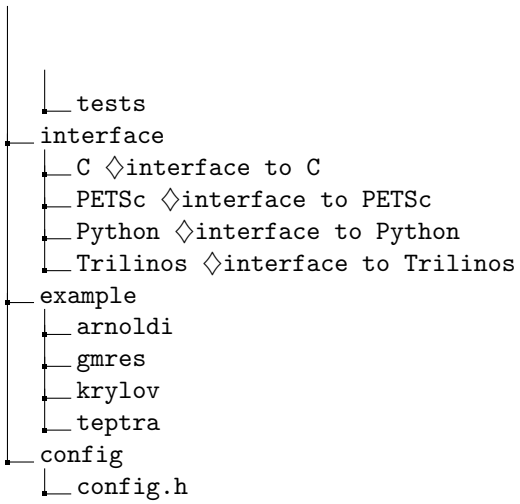
1.6 Referencing SMG2S

@article{galichergenerate, title={Generate Very Large Sparse Matrices Starting from a Given Spectrum}, author={Galicher, Hervé and Boillod-Cerneux, France and Petiton, Serge and Calvin, Christophe} }

1.7 Directory Structure

SMG2S is a collection of header files, and its directory structure is given as follows:

```
SMG2S
├── parVector
│   ├── parVectorMap.h ♢implementation of distributed vector map
│   └── parVector.h ♢implementation of distributed vector
├── parMatrix
│   ├── MatrixCSR.h ♢implementation of serial CSR Matrix
│   └── parMatrixSparse.h ♢distributed sparse matrix
├── smg2s
│   ├── specGen.h ♢Function to provide given spectrum
│   └── smg2s.h ♢ implementation of smg2s generator
├── utlis
│   ├── MPI_DataType.h
│   ├── utlis.h
│   └── logo.h
└── verification
    ├── powerIverse.cpp ♢verfication impl based on SLEPc
    └── UI ♢GUI for comparison based on SLEPc
```



1.8 List of SMG2S Contributors

This is the list of SMG2S contributors:

Xinzhe Wu	main constributor	xinzhe.wu@cea.fr
Serge Petiton	Supervisor	serge.petiton@univ-lille1.fr
Quentin Petit	GUI Implementation (Intern)	quentin.petit@polyetch-lille.net

Chapter 2

Templated SMG2S Parallel Matrix and Vector

2.1 Parallel Vector

2.1.1 Vector Map

2.1.2 Creating a Distributed Vector

```
int world_size;
int world_rank;
int span, lower_b, upper_b;
MPI_Comm_size(comm, &world_size);
MPI_Comm_rank(comm, &world_rank);
span = int(ceil(double(probSize)/double(world_size)));

if(world_rank == world_size - 1){
    lower_b = world_rank * span;
    upper_b = probSize - 1 + 1;
}else{
    lower_b = world_rank * span;
    upper_b = (world_rank + 1) * span - 1 + 1;
}

parVector<T,S> *vec = new parVector<double,int>(    \
comm,lower_b, upper_b);
```

2.1.3 Parallel Matrix

2.1.4 Matrix Map

2.1.5 Creating a Distributed Matrix

The distributed matrix should be created with the mapping of distributed vector.

```
parMatrixSparse<T,S> *A=new parMatrixSparse<T,S>(vec,vec);
```

Chapter 3

Templated Nilpotency Matrix Object

3.1 Introduction

3.2 Creating a Nilpotency Matrix Object

```
Nilpotency<int> nilp;
nilp.NilpType1(length, probSize);
```

3.3 Different Types of Nilpotency Matrix

- NilpType1;
- NilpType2;
- NilpType3;

3.4 Parameter Validation for Nilpotency Matrix

- NilpType1: parameter length can be any integer value > 0;
- NilpType2: parameter length sould be even;
- NilpType3: validation of parameter length is complex. Firstly, *length* should be divisible by p, secondly, it should exist the integers e and $f \in \mathbb{N}^*$ that makes $(d+1)e$ be divisible by $i + pf$, for all $i \in 1, 2, \cdots, \frac{d}{p}$.

Chapter 4

Generating Matrix with SMG2S

4.1 SMG2S Class

The header file ./smg2s/smg2s.h implement the matrix generation method. It is defined as:

```
template<typename T, typename S>
parMatrixSparse<T,S> *smg2s(
S probSize,
Nilpotency<S> nilp,
S lbandwidth,
std::string spectrum,
MPI_Comm comm
)
```

Inside the definition, typename T is to define the size of matrix, and typename S is to define the scalar types of entries of matrix. We give the meaning of the input parameter as below:

- **S ProbSize**: the size of matrix to generate;
- **Nilpotency<S> nilp**: the nilpotent matirx object for generation;
- **S lbandwidth**: the bandwidth of lower-diagonal band of initial matrix;
- **std::string spectrum**: the file path of spectra file;
- **MPI_Comm comm**: the working MPI communicator.

4.2 Generation Workflow

Include the head file:

```
#include <smg2s/smg2s.h>
```

Generate the Nilpotent Matrix Object:

```
Nilpotency<int> nilp;
nilp.NilpType1(length, probSize);
```

Create the parallel Sparse Matrix Object Mt:

```
parMatrixSparse<std::complex<double>,int> *Mt;
```

Generate a new matrix by SMG2S:

```
MPI_Comm comm; //working MPI Communicator
Mt = smg2s<std::complex<double>,int>(probSize, nilp,
lbandwidth, spectrum, comm);
```

Here, the **probsize** parameter represent the matrix size, **nilp** is the nilpotency matrix object that we have declared previously, **lbandwidth** is the bandwidth of lower-diagonal band. **spectrum** is the file path of spectra file, if **spectrum** is set as **" "**, SMG2S will use the mechanism inside to generate the spectral distribution. **comm** is the basic object used by MPI to determine which processes are involved in a communication.

The given spectra file is in **pseudo-Matrix Market Vector format**. For the complex eigenvalues, the given spectrum is stored in three columns, the first column is the coordinates, the second column is the real part of complex values, and the third column is the imaginary part of complex values.

```
%%MatrixMarket matrix coordinate complex general
3 3 3
1 10 6.5154
2 10.6288 3.4790
3 10.7621 5.0540
```

For the eigenvalues values, the given spectrum is stored in two columns, the first column is the coordinates, the second column is related values.

```
%%MatrixMarket matrix coordinate complex general
3 3
1 10
2 10.6288
3 10.7621
```

4.3 Creation of Given Spectrum

In the directory ./verification/tests, we give an example to generate the file of given spectrum, which can be reused by the users to create their own values. This is a small C++ file named as vecgen.cpp.

If the users want to generate the eigenvalues in time without loading from local file, they can customize their eigenvalues generation by the function specGen in the file ./verification/tests/specGen.h, and set the parameter spectrum of smg2s to be " ".

```
template<typename T, typename S>
void parVector<T,S>::specGen(std::string spectrum)
```

In this function, the eigenvalues are stored by the distributed vector text-colorblueparVector. And the filling of values on this parVector can be done by the method SetValueGlobal implemented in parVector, which takes the global indices to set values.

4.4 Customize the Low Band of Initial Matrix

We know that the low band bandwidth of initial matrix can be set by the parameter lbandwidth of smg2s. Additionaly, the distribution of entries of initial matrix can also be customized by the function matInit provided by the file ./verification/tests/specGen.h. En default, these entries are filled in random. The different mechanism to fill them will influence the sparsity of final generated sparse matrix.

```
template<typename T, typename S>
void matInit(
parMatrixSparse<T,S> *Am,
parMatrixSparse<T,S> *matAop,
S probSize,
S lbandwidth
)
```

In this function, distributed matrix *Am* and *matAop* should be filled with the same way. And these entries of matrix can be filled by the method Loc_SetValue implemented in parMatrixSparse. Loc_SetValue uses the global indices of matrix to set values.

Chapter 5

Interface to Other Languages/Libraries

SMG2S provides interfaces to C, Python, PETSc and Trilinos.

5.1 Interface to C

SMG2S install command will generate a shared library libsmg2s.so (libsmg2s2c.dylib on OS X platform) into ${INSTALL_DIRECTORY}/lib. It can be used to profit the C wrapper of SMG2S.

The way to use: 1. Add this shared library to LD_LIBRARY_PATH:

```
export LD_LIBRARY_PATH=${INSTALL_DIRECTORY}/lib
```

2. Include the header file:

```
#include <interface/C/c_wrapper.h>
```

3. create Nilpotency object :

```
struct NilpotencyInt *n;
n = newNilpotencyInt();
NilpType1(n, 2, 10);
```

4. After that, you need to create the parallel Sparse Matrix Object Mt like this :

```
struct parMatrixSparseDoubleInt *m;
m = newParMatrixSparseDoubleInt();
```

5. Generate by SMG2S :

```
smg2s(m, 10, n, 3 ," " ,MPI_COMM_WORLD);
```

6. Release Nilpotency Object and parMatrixSparse Object :

```
smg2s(m, 10, n, 3 ," " ,MPI_COMM_WORLD);
```

```
struct NilpotencyInt;
struct NilpotencyLongInt;
//double + complex + int64
struct parMatrixSparseComplexDoubleLongInt;
/*parVectorMap C wrapper*/
struct parVectorMapInt *newparVectorMapInt(void);

/*complex double long int*/

struct parMatrixSparseComplexDoubleLongInt *newPar\
MatrixSparseComplexDoubleLongInt(void);

void ReleaseParMatrixSparseComplexDoubleLongInt(struct \
parMatrixSparseComplexDoubleLongInt **ppInstance);

void LOC_MatViewComplexDoubleLongInt(struct parMatrix\
SparseComplexDoubleLongInt *m);

void GetLocalSizeComplexDoubleLongInt(struct parMatrix\
SparseComplexDoubleLongInt *m, __int64_t *rs, __int64_t *cs);

void Loc_ConvertToCSRComplexDoubleLongInt(struct parMatrix\
SparseComplexDoubleLongInt *m);

void Loc_CSRGetRowsArraySizesComplexDoubleLongInt(struct parMatri\
xSparseComplexDoubleLongInt *m, __int64_t *size,__int64_t *size2);

void Loc_CSRGetRowsArraysComplexDoubleLongInt(struct par\
MatrixSparseComplexDoubleLongInt *m, __int64_t size, int **rows,\
__int64_t size2, int **cols, double **real, double **imag);

void smg2sComplexDoubleLongInt(struct parMatrixSparseComplex\
DoubleLongInt *m, __int64_t probSize, struct NilpotencyLongInt \
*nilp, __int64_t lbandwidth, char *spectrum, MPI_Comm comm);
```

5.2 Interface to Python

SMG2S uses SWIG to generate the wrapper of SMG2S to Python. Generate the shared library and install the python module of smg2s.

```
cd ./interface/python;
mpicxx -fpic -c smg2s_wrap.cxx  -I/apps/python/3/    \
include/python3.5m -std=c++0x
mpicxx -shared smg2s_wrap.o -o _smg2s.so
python setup.py install
```

Before the utilisation, make sure that **mpi4py** is installed.
This is a little example of usge :

```
from mpi4py import MPI
import smg2s
import sys

size = MPI.COMM_WORLD.Get_size()
rank = MPI.COMM_WORLD.Get_rank()
name = MPI.Get_processor_name()
```

```
sys.stdout.write(
"Hello, World! I am process %d of %d on %s.\n"
% (rank, size, name))

if rank == 0:
    print ('INFO ]> Starting ...')
    print("INFO ]> The MPI World Size is %d" %size)

#bandwidth for the lower band of initial matrix
lbandwidth = 3

#create the nilpotent matrix
nilp = smg2s.NilpotencyInt()

#setup the nilpotent matrix:
nilp.NilpType1(2,10)

Mt = smg2s.parMatrixSparseDoubleInt()

#Generate Mt by SMG2S
#vector.txt is the file that stores the given    \
spectral distribution in local filesystem.
Mt=smg2s.smg2sDoubleInt(10,nilp,lbandwidth,     \
"vector.txt", MPI.COMM_WORLD)
```

5.3 Interface to PETSc

SMG2S provides the interface to scientific computational softwares PETSc/SLEPc.

The way of Usage:

Include header file: Include the header file:

```
#include <interface/PETSc/petsc_interface.h>
```

Create parMatrixSparse type matrix :

```
parMatrixSparse<std::complex<double>,int> *Mt;
```

Restore this matrix into CSR format :

```
Mt->Loc_ConvertToCSR();
```

Create PETSc MAT type :

```
MatCreate(PETSC_COMM_WORLD,&A);
```

Convert to PETSc MAT format :

Create PETSc MAT type :

```
A = ConvertToPETSCMat(Mt);
```

Here are the example of Arnoldi, GMRES, and another Krylov method.

5.4 Interface to Trilinos/Teptra

SMG2S is able to convert its distributed to the CSR one-dimensional distributed matrix defined by Teptra in Trilinos.

The way of usage:

Include header file

```
#include <interface/Trilinos/trilinos_interface.hpp>
```

Create parMatrixSparse type matrix :

```
parMatrixSparse<std::complex<double>,int> *Mt;
```

Create Trilinos/Teptra MAT type :

```
parMatrixSparse<std::complex<double>,int> *Mt;
```

Convert to Trilinos MAT format :

```
K = ConvertToTrilinosMat(Mt);
```

Here is a full example of Trilinos.

Chapter 6

Verification of Eigenvalues

SMG2S provides the functionality to verify the abilibity to keep given spectrum. In the directory of **./verification/**. The implementation of the functionality is powerInverse.cpp.

6.1 Prerequisites

The verfication method is implemented based on the shifted inverse method proposed by PETSc/SLEPc. Before the starting of verification, it is necessary to have the two on the platforms.

If not, the download and installation of PETSc can be found: [PETSc Download] and [SLPEc Installation]. The download and installation of PETSc can be found: [SLEPc Download] and [SLEPc Installation].

6.2 Verifcation by Shifted Inverse Power Method

1. compile the file powerInverse.cpp by the command

```
make
```

This will generate an executable powerInverse.exe.

2. Suppose the given eigenvalues are stored in the file vector.txt by the pseudo-Matrix Matrix Vector format, run the verification script as blow:

```
#!/bin/bash
EXEC=./powerInverse.exe
N=100
L=10
TEST_TOL=0.00001
DEGREE=4

LENGTH=$(awk 'NR==2{print $1}' vector.txt)
echo "Test Eigenvalues number = "${LENGTH}

for((i=3;i<=${LENGTH}+2;i++))
do
```

```
real=$(awk 'NR=='$i'{print $2}' vector.txt)
imag=$(awk 'NR=='$i'{print $3}' vector.txt)
srun -n 1 ${EXEC} -n ${N} -l ${L} -eps_monitor_conv    \
-eps_power_shift_type constant -st_type sinvert      \
-exact_value ${real}+${imag}i -test_tol ${TEST_TOL}    \
-degree ${DEGREE}
done
```

Here we list the meaning of the important parameters in the script above:

- N: the size of matrix to generate, which should be equal to the number of given eigenvalues;
- L: the bandwidth of low part diagonal of matrix to generate;
- TEST_TOL: the tolerance to check if the accuracy of one eigenvalue can be accepted or not;
- DEGREE: the continuous one for the nilpotency matrix.

6.3 Script for result cleaning

The result file generated during the verification can be cleaned into the pseudo-Matrix Market Vector by the script below:

```
#!/bin/bash

grep "@> The eigenvalue" $1 > tmp.txt
awk '{print $5 " " $7 }' tmp.txt > tmp2.txt
awk '{print substr($0, 1, length($0)-1)}' tmp2.txt \
> tmp3.txt

awk '{print NR  " " $0}' tmp3.txt > tmp4.txt
NB=`wc -l tmp4.txt | awk '{print $1}'`
awk 'BEGIN{print '$NB' " " '$NB' " " '$NB'}{print}' \
tmp4.txt  > tmp5.txt

awk 'BEGIN{print "%%MatrixMarket matrix coordinate \
 real general"}{print}' tmp5.txt   > $2

rm tmp.txt tmp2.txt tmp3.txt tmp4.txt tmp5.txt
```

Execution of this script:

```
./traitement.sh results.txt results_clean.txt
```

In this command, The 1st and 2nd arugments for the execution are seperately the initial results file and the final cleaned and formatted file.

6.4 Plot by Graphic User Interface

6.4.1 Prerequisites for GUI

You need to have Python2.X or Python3.X to run it. Moreover, UI uses some libraries to support a dynamic and intuitive graphical user interface, you can see the list of libraries. Normally, some of them are include in Python distribution. You can find below the list of necessary libraries of the UI.

- Modules which are bundled in the Python installation : Tkinter, re, sys, decimal ;
- Modules which need to be installed in addition to Python : NumPy & SciPy, Matplotlib, Pillow(PIL)

Install modules to Python 2.X:

```
apt-get install python-tk python-imaging-tk
pip -m install Pillow
python -mpip install -U pip
python -mpip install -U matplotlib
pip install -U numpy scipy
```

Install modules to Python 3.X:

```
sudo apt-get install python3-tk python-imaging-tk
pip -m install Pillow
python -mpip install -U pip
python -mpip install -U matplotlib
pip install -U numpy scipy
```

6.4.2 How to use the GUI

To use the GUI:

```
python main.py
```

When you launch the program, a new windows opens like Fig. 6.4.2 :

The first step is to select the files which be display. When you have selected a file, the button change in green color as Fig. 6.4.2 (Attention, the files imported should be in the pseudo-Matrix Market vector format that we have talked):

After that, you can click on "Display" to build and open the graphic on the right side of the window. Click on "New window" to open your graphic on a new window. It's possible to open as many windows as you want like Fig. 6.4.2:

Your will be generate with automatic lens scaling, but your can generate it with your own scales as Fig. 6.4.2:

Figure 6.1: Home Screen Capture

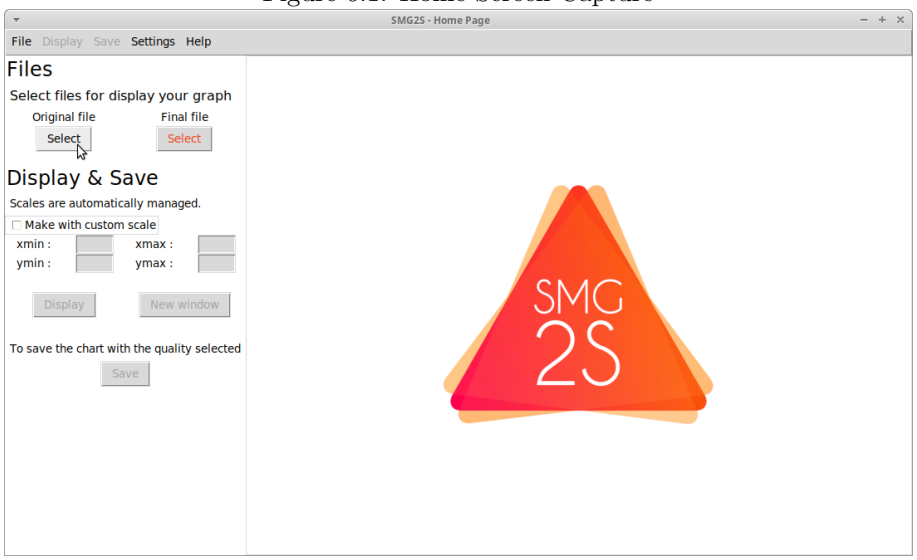


Figure 6.2: Home Screen Capture After Selection

Figure 6.3: Home Screen Plot Capture

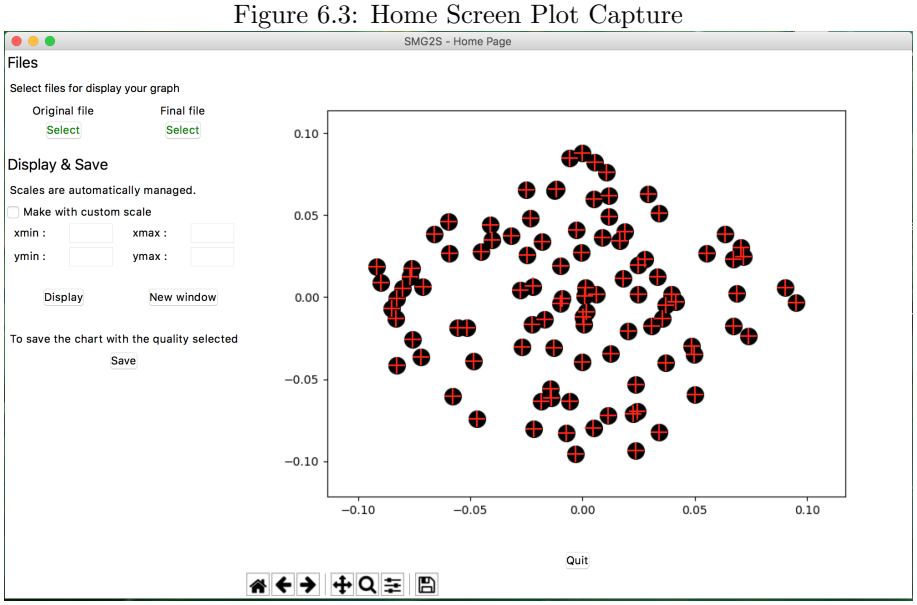


Figure 6.4: Home Screen Plot Capture with Lens Scaling