SuperStiff documentation

March 25, 2019

Contents

1 Outlook

This documentation is addressed to persons that intend to compute the superfluid stiffness using Cluster Dynamical Mean Field Theory [1]. The main programs have been tailored for the usage of 8×8 cluster self-energies Σ_c, that is self-energies obtained solving a 2×2 cluster impurity. It can be straightforwardly adapted for bigger clusters.

In this document, the following aspects are set out:

- The main tasks SuperStiff accomplishes
- The structure of the program SuperStiff
- The main functions of the different modules

I will also follow through different examples given in the folder `examples`. Some of the examples are results already shown in my master's thesis [2]. If you still haven't read the main `README` file stored in the folder `SuperStiff`, take a look! This document gives complementary informations for anyone that wants to dig in deeper.

The calculations can be made on a personal laptop since the program is not too time-consuming. This Julia program has only been benchmarked for Linux operating systems. Nervertheless, the installations of Julia and the different packages are quite easy and do not depend on the operating system as far as I can tell[1].

2 Purpose of the program

The main purpose of SuperStiff is to compute the superfluid stiffness in the layered cuprates. It is a wrapper over a CDMFT procedure that would provide converged self-energies. This program computes the superfluid stiffness along all the principal axes of the unit-cell. The current vertex corrections are neglected. It computes the superfluid stiffness using any of the periodization schemes available, that is, periodizing the Green's function (*pér.* G), periodizing the cumulant of the Green's function (*pér.* M), or tracing over the cluster Green's function (*tr.*) (see section 4.2.3 of Ref.[2]). It does so if ones has a self-energy that has converged in the pure state or in the mixed state AF+SC.

3 Structure of the program

The structure of the program is glanced over in this section, setting out the necessary informations to provide in order for the program to successfully run. The program uses the Julia programming language. Versions of the code are available in both Python and C++, but these versions have had poor maintenance and are prone to bugs.
The program is composed of three different modules stored in folder `src`, three different main files stored in the folder `examples` and one input file named `params.json`. The objective here is to explain each and every entry in the input file. The content of the `params.json` file is exposed in listing 1. It can be helpful to read first off the appendices M and N of Ref.[2].

[1]Julia programming language is well supported for any operating systems (MacOs, Windows or Linux)

3.1 Zero temperature calculation

For zero temperature calculations of the superfluid stiffness, the main program `OutofPlaneSuperStiff.jl` can be used to compute the c-axis current-current correlation function or `InplaneSuperStiff.jl` can be used to compute the a- or b-axis current-current correlation function. Each of the these main programs needs the input file `params.json` to specify important parameters. The content of the `params.json` file is described below.

```json
{
    "t": 1.0,
    "tp": -0.3,
    "tpp": 0.2,
    "inplane_axis": "xx",
    "path_to_files": "COEX/U8/SEvec_b500_SC_AFM/",
    "data_loop": "COEX/U8/Loop_COEX.dat",
    "AFM_SC_NOCOEX": 0,
    "Print_mu_dop": 0,
    "pattern": "SEvec*.npy",
    "beta": 500,
    "w_discretization": 2000,
    "cumulant": 0,
    "Periodization": 1,
    "fout_name": "stiffness_b500_w_2000_coex_int_K_per_U8_xx.dat"
}
```

Listing 1: Content of params.json

The first three parameters stand for three nearest-neighbor tight-binding hopping terms: t depicts the nearest-neighbor hopping term, t' the second nearest-neighbor hopping term and t'' the third nearest-neighbor hopping term. The example given is for YBCO. The other input parameters are

"inplane axis": specifies the axis along which the superfluid stiffness is to be computed. Only the following set of strings is acceptable: {"xx","yy"}. This parameter doesn't matter if the c-axis superfluid stiffness is computed (zz).

"path_to_files": specifies the file path containing the cluster self-energies resolved in Matsubara frequencies. The file path MUST start by either "NOCOEX" in the case of pure SC calculations or "COEX" in the case of mixed AF+SC calculations.

"data_loop": specifies the file path containing the chemical potential, the particle density and the order parameter amplitude(s). The number of self-energy binary files (*.npy) has to be the same as the number of lines in the `NOCOEX/U8/Loop_NOCOEX.dat` file and the integer specifying the line has to correspond to the one of the self-energy binary plus 2 (because Python and C++ starts with number 0 and there is a header). For example, the binary file named `NOCOEX/U8/SEvec_b500_SC/SEvec_b500_SC138.npy` corresponds to line 140 in the file `NOCOEX/U8/Loop_NOCOEX.dat`.

"AFM_SC_NOCOEX": this field takes in the binary set of values $\{0, 1\}$. For example, when 0 is entered, the formula Eq.(5.31) or Eq.(5.37) of Ref.[2] are used with the CDMFT self-energy converged in the mixed state in the case `OutofPlaneSuperStiff.jl` is launched. The formula Eq.(5.31) is used if **"Periodization"** is set to 0 and Eq.(5.37) if 1 is provided instead. If the file `InplaneSuperStiff.jl` were used instead and **"AFM_SC_NOCOEX"** were set to 0, it would have called Eq.(5.30) having set **"Periodization"** to 0 or Eq.(5.36) having set **"Periodization"**

to 1. If otherwise **"AFM_SC_NOCOEX"** is set to 1, both the input fields **"path_to_files"** and **"data_loop"** have to start with NOCOEX in order to use the superfluid formulae developed in the regime of AF+SC coexistence with CDMFT data converged in the pure SC state. Most of the time, **"AFM_SC_NOCOEX"** is set to 0.

"Print_mu_dop": this field is only relevant when debugging and takes in $\{0, 1\}$. This entry is useful when SuperStiff is used in conjonction with other programs that are kept private. Always set to 0. Prints out some information and the program might not work if set to 1.

"pattern": specifies the pattern of the binary files contained in the path **"path_to_files"**. It can be whatever string value, as long as it is labelled with an integer as mentionned previously and it is a binary file.

"beta": gives the fictitious temperature that is used to sum over the fermionic Matsubara frequencies. The value of **"beta"** can be changed, but it is preferable to keep its value to 500. The higher the value is, the better it is if one periodizes the Green's function (it is not the case if one periodizes the cumulant or traces), but the calculations are lengthened. Setting it to 500 is the best compromise I have found.

"w_discretization": gives the number of fermionic Matsubara frequencies that compose the grid upon which one sums over. The value of 2000 can reduced, as the important is to have a great resolution at low frequencies (the superfluid stiffness converges as $\propto \frac{1}{(i\omega_n)^4}$).

"cumulant": specifies if the cumulant of the Green's function is to be periodized: when its value is set to 0 and **"Periodization"** is set to 1, the Green's function is periodized. If its value is set to 1 and **"Periodization"** is set to 1, the cumulant is instead periodized. To trace over the cluster Green's function in order to avoid any periodization, one has to set both **"cumulant"** and **"Periodization"** to 0. Notice that if **"Periodization"** is set to 0, **"cumulant"** has no effect whatsoever.

"Periodization": has already been talked about quite a lot. This field takes in the values $\{0, 1\}$. If 0 is chosen, then the cluster Green's function is not periodized when computing the superfluid stiffness. Otherwise, if 1 is chosen, the cluster Green's function is periodized, either the cumulant or the Green's function itself, depending on the value of **"cumulant"**.

"fout_name": is the string name of the file that will contain the superfluid stiffness. One can name as he/she wants. The values are appended dynamically in the file at runtime. Interrupting the program does not erase the progress of the program.

3.2 Finite temperature calculation

To perform finite temperature calculations, no `params.json` input file is needed. All the important parameters to feed in are specified in the main program itself, that is the `FiniteTSuperstiff.jl` file.

```julia
using SuperStiff.PeriodizeSC
using NPZ
using Glob

# You have to run convert_converged_green() before running this program
   properly (produce both selfR and greenR)

filename_to_write = "PER_periodized_zz.dat"

t = 1.0; tpp = 0.0
pwd_ = pwd()
```

```
path_ = pwd_*"/"*filename_to_write
Grid_ = 100 # Resolution of k-space that is relevant ONLY for inplane
    calculations
OPT_ = "PER"  # Options are CUM, PER and TR
AXIS_ = "zz" # Options are xx (a axis), yy (b axis) and zz (c axis)
```

Listing 2: Input parameters in FiniteTSuperstiff.jl

The listing 2 shows the input parameters necessary for finite temperature calculations and each of these are explained and detailed below:

filename_to_write: sets the name of the output file produced by the main program `FiniteTSuperstiff.jl`

t: specifies the nearest-neighbor hopping term. It should always be set to 1.0, as it represents the energy scale of the system.

tpp: specifies the third nearest-neighbor hopping term. It should be set to 0.0 if one only considers the nearest-neighbor and second nearest-neighbor hopping term, as is our case here.

Grid_: determines the resolution of the **k**-space grid when the parameter **AXIS_** is set to "xx" or "yy". To set it to 100 is a sensible choice.

OPT_: specifies the periodization scheme ("PER" or "CUM") to calculate the superfluid stiffness. If one wants to calculate the superfluid stiffness by tracing over the cluster Green's function, one has to type in "TR". If "TR" is chosen, one has to set **AXIS_** to "zz".

AXIS_: specifies the principal axis of the unit-cell along which the superfluid stiffness is to be computed. The following set holds all the permissible input parameters: {"xx","yy","zz"}.

The example given in the folder `examples` concern the case where only the second nearest-neighbor hopping term is changed (t'). The folder structure given in this example MUST be followed for the program to succeed. Inside the folder specifying the value of t', one has to name the folder containing the raw cluster self-energies computed at different temperatures in the following way:

$$\text{U}\underbrace{8}_{\substack{\text{value of}\\\text{Hubbard interaction}}}\text{m}\underbrace{5.65}_{\substack{\text{value of}\\\text{chemical potential}}}\underbrace{\text{\_all.beta}}_{\substack{\text{contains}\\\text{all}\\\text{temperature calculations}}}. \tag{1}$$

If for a given temperature the cluster self-energies don't show any anomalous components, the folder containing the self-energies at each iteration MUST end by "n". This way, these folders are ignored by the program. The main program `FiniteTSuperstiff.jl` is a wrapper to the program `Lazyskiplist`.

3.3 Modules

In this subsection, I give some broad information about the modules called by the main programs, althought it is somewhat straightforward. Some useful informations are provided in case anyone were to extend the utility/scope of this program. The three following modules are necessary for SuperStiff to run:

- SuperStiff.jl
- PeriodizeSC.jl
- Stiffness.jl

The first module SuperStiff is the module that pre-compiles the two other modules, that is PeriodizeSC and Stiffness (it acts as the binder). All these files have to be stored as indicated in the `INSTALL` file in `src`.

The second module PeriodizeSC is the module containing the lower level functions. It contains the function that build the cluster Green's function from the cluster self-energy. It contains all the different superfluid stiffness formulae. It also contains the many wrapper functions (decorators) necessary to have to integrate using cubature. It is the heaviest module of SuperStiff.

The last module Stiffness calls in PeriodizeSC — it builds on its member functions (inherits from it). This module loads the `params.json` file holding the instructions to the program. From the information taken from the input file, it selects the proper functions to be called from PeriodizeSC. If anyone modifies the input parameters of `params.json`, one has to cast his/her attention particularly on the Stiffness module.

From this small overview and the several examples provided in the folder `examples`, one should be able to take ownership of this program.

References

[1] M. Charlebois, *Théorie de champ moyen dynamique pour les systèmes inhomogènes*. Phd thesis, Université de Sherbrooke, Sherbrooke, 2015.

[2] O. Simard, "Rigidité superfluide et température critique en présence d'une autre phase," Master's thesis, Université de Sherbrooke, Sherbrooke, 3 2019. Link: http://hdl.handle.net/11143/15039.