# mfsw_test

`mfsw_test` is a test directory for running a program, `test.cpp`, built by applying the mean-field theory and spin-wave theory to the general localized interaction model, $$ \mathcal{H} \ = \ \sum_{\braket{ij}} \sum_{\xi\xi^\prime} I_{ij}^{\xi\xi^\prime} \mathcal{O}_i^{\xi} \mathcal{O}_j^{\xi^\prime}

- \sum_{i}\sum_{\xi} H_i^\xi \mathcal{O}_{i}^\xi \ = \ \mathcal{H}^{\text{MF}} + \mathcal{H}^\prime. $$

One can calculate as following quantities.

- Local state
- Spin-wave dispersion
- Dynamical structure factor $\mathrm{S}(\mathbf{q},\omega)$
- Moment contraction (or SW moment)
- Quasiparticle damping rate

As for the dynamical structure factor, in the linear spin-wave theory, $\mathrm{S}_{\xi\xi'}(\mathbf{q},\omega)$ is expressed by $$ S_{\xi\xi^\prime} (\mathbf{q},\omega) \ = \sum_{\eta}^{N} \tilde{W}{|eta,|mathbf{q}}^{|xi} |tilde{W}{\eta,\mathbf{q}}^{\xi^\prime*} \delta (\omega - \varepsilon_{\eta,\mathbf{q}}), $$ where, N is a number of spin-wave bands and $\varepsilon_{\eta,\mathbf{q}}$ is the excitation energy for the $\eta$-th band.

On the program, all $\mathrm{W}^{\xi}_{\eta,\mathbf{q}}$ are calculated and written to a file opened by `std::ofstream`.

# Overview

Below is the code to calculate the mean-field solution, spin-wave dispersion, SW moments, and dynamical structure factor.

```
#include <iostream>
#include <fstream>
#include <complex>
#include <string>
#include "cpplapack/cpplapack.h"

using namespace std;


#include "mfsw.hpp"
#include "makedata.hpp"

using exch_type = CPPL::dgematrix;
struct param
{
  vector<tuple<vector<double>, exch_type>> J;
  vector<vector<double>> H;
```

```
  double T;
};

void bond(int n, int Nc, int i, int j, const vector<double> &xs,
mfsw<exch_type>& ms,
          param &p)
{
  int m;
  exch_type Js;
  for (int i = 0; i < p.J.size(); i++)
  {
    vector<double> x = std::get<0>(p.J[i]);
    double sum = 0.;
    for (int k = 0; k < xs.size(); k++)
      sum += abs(x[k] - xs[k]);
    double eps = 1e-4;
    if (sum < eps)
    {
      m = i;
      Js = std::get<1>(p.J[m]);
      break;
    }
    else
    {
      Js.resize(Nc, Nc); Js.zero();
    }
  }
  ms.set_J(n) = std::tuple<int, int, exch_type>{i, j, Js};
}


int main(int argc, char *argv[])
{
  if(argc!=1)
    exit(1);

  std::string fn_lattice =
"data/input/spin_wave_230203/cubic2subbcc_lattice.dat";
  std::string fn_iij =
"data/input/spin_wave_230203/cubic2subbcc_iij_1_1.dat";
  std::string fn_base =
"data/input/spin_wave_230203/cubic2subbcc_base_1_1.dat";

  makedata<exch_type> md(fn_lattice, fn_iij, fn_base);

  complex<double> im(0, 1);

  int Ns = md.make_Ns();       // 局所状態の数  (generator の次元)
  int Nc = md.make_Nc();       // サイトあたりの平均場の数
  int M = 2;                   // 副格子の数
  int N = M * Nc;              // 平均場の数
  int Z = 8;                   // １つの注目サイトと相互作用するサイトの合計（最近
接間でのみ相互作用する bcc なら  8）
  int Nb = M * Z/2;            // ユニットセルに含まれるボンドの総数
```

```
  int Nr = 5;                    // 初期値を（ランダムに）入力する回数
  int Ni = 100000;               // ループ回数



  ////*generator*////
  vector<CPPL::zhematrix> Oop(Nc);   // generator
  for (int i = 0; i < Oop.size(); i++)
    Oop[i] = md.make_generator()[i];


  param p;
  md.make_J(p.J);
  md.make_H(p.H, M);

  mfsw<exch_type> ms(N, Nc, Nb, Ni, Nr, 1e-8);

  for (size_t i = 0; i < N; i += Nc)
  {
    for (size_t j = 0; j < Nc; j++)
      ms.set_mat(i + j) = Oop[j];
  }

  for(size_t i = 0; i < N; i += Nc)
  {
    for (size_t j = 0; j < Nc; j++)
    {
      ms.set_H(i + j) = p.H[i / Nc][j];
    }
  }


  vector<vector<double>> xs(Nb);   // ユニットセル内の独立なボンド間の相対座標

  /* bond input */
  {
    ms.set_bond(xs, Nb, md);       // 元データと同じ副格子構造を仮定していれば
md.set_bond を呼び出すだけで良い
  }

  ms.exec_mf();                    // Nr 回のランダムな初期値で解を探す
  cout << "# ";
  cout << ms.mf_out() << endl;     // 平均場解の出力（Nr回実行した内の最安定解のみ
出力される）




  p.T = 1e-4;                      // System の温度
  ms.set_T() = p.T;
  std::function<complex<double>(vector<double>, vector<double>)> g_;
  g_ = [&im](vector<double> x, vector<double> k)
  {
    complex<double> pd(1);
```

```
    for (int i = 0; i < x.size(); i++)
      pd = pd * exp(im * x[i] * k[i]);

    return pd;
  };

  vector<std::function<complex<double>(vector<double>, vector<double>)>>
g(Nb);
  for (size_t i = 0; i < Nb; i++)
    g[i] = g_;

  vector<vector<double>> a(3);         // 基本並進ベクトル
  md.make_unitvec(a);

  std::string fnsw = "data/output/spec.txt";
  std::ofstream sw(fnsw);

  {
    double x = 0.;
    double t = 0.;

    /* (-2π,-2π,-2π) --- (2π,2π,2π) line */
    for (x = -2.; x < 2.001; x += 0.005)
    {
      vector<double> k(3);
      k[0] = x;
      k[1] = x;
      k[2] = x;
      k[0] *= M_PI;
      k[1] *= M_PI;
      k[2] *= M_PI;
      vector<complex<double>> gamma(g.size());
      for (int i = 0; i < Nb; i++)
        gamma[i] = g[i](xs[i], k);
      sw << x << " " << ms.exec_sw_out(gamma);   // スピン波エネルギーおよび動
的構造因子の書き込み
      sw << endl;
    }
    t = x;
  }

  std::string fnmg =  "data/output/swmag.txt";
  std::ofstream mg(fnmg);
  {
    double dk = 0.05;
    mg << " " << ms.exec_sw_mag_out(xs, g, a, dk) << endl;  // SW モーメント
の書き込み
  }

}
```

Note the following:

1. `cout << ms.mf_out() << endl;` outputs in the following order:

$$ \begin{align} &E_{\text{gs}}, \nonumber\ &\braket{\mathcal{O}^{\xi=0}}{A}, | |braket{|mathcal{O}^{|xi=1}}{A}, \cdots, \ \braket{\mathcal{O}^{\xi=\xi_{\text{max}}}}{A},| |braket{|mathcal{O}^{|xi=0}}{B}, \ \cdots, \ \braket{\mathcal{O}^{\xi=\xi_{\text{max}}}}{B}, |braket{|mathcal{O}^{|xi=0}}{C}, \ \cdots \nonumber \end{align} $$ where, capital alphabets represent sublattices. As an example, for the data of `"cubic2subbcc"`, the output is as follows.

```
#  -1.00000e+00
0.7071067973 0.5845336644 -0.0056875930 -0.3978543406 0.7071066436
-0.5845336470 0.0056875929 0.3978543288
```

2. `sw << x << " " << ms.exec_sw_out(gamma);` write out in `fnsw` in the following order:

$$ \begin{align} &x, \varepsilon_{0,\mathbf{q}},\ \varepsilon_{1,\mathbf{q}}, \cdots, \varepsilon_{N,\mathbf{q}},\ \Big|\tilde{W}{0,|mathbf{q}}^{|xi=0}|Big|^2, |Big|tilde{W}{1,\mathbf{q}}^{\xi=0}\Big|^2 ,\cdots ,\Big|\tilde{W}{N,|mathbf{q}}^{|xi=0}|Big|^2,| |Big|tilde{W}{0,\mathbf{q}}^{\xi=1}\Big|^2, \Big|\tilde{W}{1,|mathbf{q}}^{|xi=1}|Big|^2 ,|cdots ,|Big|tilde{W}{N,\mathbf{q}}^{\xi=1}\Big|^2,\ \cdots\cdots\cdots\cdots, \Big|\tilde{W}{0,|mathbf{q}}^{|xi=|xi{\text{max}}}}\Big|^2, \Big|\tilde{W}{1,|mathbf{q}}^{|xi=|xi{\text{max}}}}\Big|^2 ,\cdots ,\Big|\tilde{W}{N,|mathbf{q}}^{|xi=|xi{\text{max}}}}\Big|^2 \nonumber\ & ,\text{Re} (\text{off-diagonal terms}),\ \text{Im}(\text{off-diagonal terms}) \nonumber \end{align} $$

3. `mg << " " << ms.exec_sw_mag_out(xs, g, a, dk) << endl;` write out in `fnmg` in the following order:

$$ \newcommand{\bbraket}[1]{\langle\langle#1\rangle\rangle} \begin{align} &\bbraket{\mathcal{O}^{\xi=0}}{A}, | |bbraket{|mathcal{O}^{|xi=1}}{A}, \cdots, \ \bbraket{\mathcal{O}^{\xi=\xi_{\text{max}}}}{A},| |bbraket{|mathcal{O}^{|xi=0}}{B}, \ \cdots, \ \bbraket{\mathcal{O}^{\xi=\xi_{\text{max}}}}{B}, |bbraket{|mathcal{O}^{|xi=0}}{C}, \ \cdots \nonumber \end{align} $$ where, $\langle\langle ~\cdot~ \rangle\rangle$ is the expectation value in Bogoliubov vacuum.

# Interfaces

`makedata.hpp` and `mfsw.hpp` provide one interface each.

・makedata

The primary role of `makedata` class is to produce $\mathcal{O}*i,*{ij},H_{i}$, and the primitive lattice vector from the original data.

・mfsw

'mfsw' class computes the mean-field solution and above quauntities. In the mean-field solution, we can change the initial values to file read.

For example, we prepare the initial value input file

```
0.7071067812 0 0 0.7071067812  0.7071067812 0 0 -0.7071067812
```

and change `ms.exec_mf()` to the following.

```
ms.exec_mf("data/input/init.txt");
```

As a result, the output is as follows.

```
#  -1.00000e+00
0.7071067812 0.0000000000 0.0000000000 0.7071067812 0.7071067812
0.0000000000 0.0000000000 -0.7071067812
```

Note that if you put some initial values, it will solve a self-consistent equation for the number of initial values.

# Graphics

The spin-wave dispersion and dynamical structure factor are plotted by `sw.py` and `spec.py`. These python codes use command-line arguments. Run these codes as follows:

```
$ python sw.py M Ne
$ python spec.py M Ne
```

Here, M is total number of sublattices and Ne is total number of local excited states. For `"cubic2subbcc"`, we type as

```
$ python sw.py 2 1
$ python spec.py 2 1
```

# Requirement

- C++11 compatible environment
- cpplapack-2015.05.11 (header-only library)