Chapter 1

Introduction

The package *colloid* 0.9 is used to analyze the experimental data obtained from colloidal systems.

This package requires those packages to be installed first: BLAS, LAPACK, FFTW. Make sure unistd.h is available.

The data is passed as arrays, and saved and read through *gdf* format. The detail format information can be seen in io.h and io.cpp, and the core of data passing is done by classes and functions in colloid_base.h and colloid_base.cpp. All program should include them as the data I/O and data type.

Some statistics functions is in *statistics*. A few functions are added in *miscellaneous*, such as sort, fitting. FFT is in a separate place so that one can able/disable FFT easily. Image analysis part is so far based on Magick++.

All other codes can be seen as packages as the extension of the core. Different packages can be added/removed without affecting others. Those are normal modes, or elasticity.

Some functions has been documented and the documents can be found in the corresponding header file.

Linux command line interfaces are stored in functions/. Run the batch_compile and make sure the link in compile_and_run is correct.

Chapter 2

Normal mode

2.1 Algorithm

The algorithm for calculating normal modes are the displacement covariance matrix: $\langle u_i u_j \rangle = k_B T [K^{-1}]_{ij}$.

To plot the modes, Gnuplot is used.

To calculate level spacing distribution and spectral rigidity, refer to the random matrix theory:

```
@BOOK{mehta04,
author = "M. L. Mehta",
title = "Random Matrices",
edition = "3rd ed.",
publisher = "Academic Press, New York",
year = "2004"
}
```

The transverse and longitudinal part, dispersion relations are calculated according to their definition. From the dispersion relation, one can obtain sound velocity $v = \lim_{k\to 0} \omega/k$, and the elastic modulus $E = v^2 \rho$.

2.2 Binary file format

```
==================
FORMAT of .mp FILE
==================
header0 header1 mp

header0: int, 'm'*256+'p'
```

```
header1: int, numbers of points * 2
mp[header1] : float, xy positions
mp[2*i]   : x position of point i
mp[2*i+1] : y position of point i

==================
FORMAT of .ev FILE
==================
header0 header1 E G

header0: int, 'e'*256+'v'
header1: int, numbers of eigenvalues
E[header1] : double, eigenvalues
G[header1*header1]  : double, eigenvectors.
G[i*header+j] is eigenvectors corresponding eigenvalue E[i].
```

2.3 Codes

Binargy I/O

- write and read .ev file: writeev, readev
- read only a mode from .ev file: readmode
- read only eigenvalues from .ev file: readE
- write and read .dcm file (dynamic covariance matrix): writedcm, readdcm, writedcm3D, readdcm3D
- write and read .mp file (mean position): writemp, readmp

Core functions:

```
/*! normal mode core
 *  @param ptid
 *   (type)    colloid_base &
 *   (input)   [position, ...,  time, id] data
 *  @param file
 *   (type)    const char *
 *   (input)   filename prefix (can contain subfix) to store eigenvalues
 *             and eigenvectors of covariance matrix
 *  @param Remove_Drift
 *   (type)    bool
 *   (input)   remove drift or not
 *   (default) remove drift
 *
 *  (input)ptid should contain same number of particles for each frame.
 *  .ev file with prefix (input)file is generated, to store eigenvalues and
 *  eigenvectors of covariance matrix.
 */
void normal_mode(colloid_base&, const char *, double =-1, bool =true);


/*! density of states core
 *  @param Gn
 *   (type)    int
 *   (input)   number of eigenvalues
 *  @param E
 *   (type)    double *
 *   (input)   eigenvalues
 *  @param file
 *   (type)    const char *
 *   (input)   filename prefix (can contain subfix) for output
 *
 *  The output is with a subfix ".dos"
 *  This program is called in
 *   normal_mode
 *   DOS
 */
void DOS_w(int, double *, const char *);


/*! density of states
 *  @param ptid
 *   (type)    colloid_base &
 *   (input)   [position, ...,  time, id] data
 *  @param file
 *   (type)    const char *
 *   (input)   filename prefix (can contain subfix) for output
 *  @param Remove_Drift
 *   (type)    bool
 *   (input)   remove drift or not
 *   (default) remove drift
 *
 *  (input)ptid should contain same number of particles for each frame.
 *  The program first try to read eigen data from .ev file if found, else
 *  it will call normal_mode to generate everything.
 */
void DOS(colloid_base &, const char *, bool);

/* logarithmic binning of DOS */
void DOSlog(const char *);
void DOS_log(const char *file);

/* DOS obtatined by kernel density estimate */
```

```
void DOS_kdensity(const char *file, double h, int nbin);

/* participation ratio and smoothing */
void participation_ratio_w(int, double *, double *, const char *);
void participation_ratio(colloid_base &, const char *, bool);
void participation_ratio_smooth(const char *file, double h);

/* level sapcing distribution and spectral rigidity */
void level_separation_w(int, double *, const char *);
void level_separation(colloid_base &, const char *, bool);
void level_separation_log(const char *);
void level_spacing(const char *file, double h, unfolding_t UNFOLDING);
void level_rigidity(const char *file, double h, unfolding_t UNFOLDING);

/* amplitude distribution of a given mode */
void mode_hist_w(int, double *, int, const char *);
void mode_hist(colloid_base &, int, const char *, bool);

/* obtain covariance matrix */
void covariance(colloid_base &, const char *, bool =true);

/* spatial displacement distribution */
void uur(colloid_base &, const char *, double =-1, bool =true);

/* decomposition of a mode to transverse and longitudinal part */
void transverse_longitudinal(int&, const char *);
void transverse_longitudinal_eps(int&, const char *);

/* plot modes and ouput eps file */
void plot_mode(int&, const char *);
void plot_mode_eps(int&, const char *);

/*! project displacement to normal modes */
void projection(const char *, double =-1, bool =true);


/*! dispersion relation of the system */
void dispersion(const char *);
void plot_dispersion(const char *, int =0);

/*! inverse of dymamic covariance matrix,
 * the hessian matrix of the system
 */
void K(const char *);
void Kr(const char *);

/*! binary IO of K */
void writeK(int &, double *, const char *);
void readK(int &, double **, const char *);
```

```
/* plot K */
void plot_K(const char *file);

/*! potential energy of each frame
 * should pass the same argument with normal_mode
 */
void U(const char *, double =-1, bool =true);

/* soft spots */
void softspot(const char *file, int Nmode, int Nsmall);

/* spatial correlation of amplitudes of a mode */
void mode_spcorr(int n, const char *file);
void mode_spcorr_all(const char *file, double omegaRange);
```

Chapter 3

Package: ellipsoid_normalmode

3.1 Theory

The Langevin's equation for a 1D Brownian particle diffusing in potential well $U(x) = Kx^2/2$ can be written as

$$M\ddot{x} = -Kx - \gamma\dot{x} + \eta, \tag{3.1}$$

where η is the white noise, satisfying $\langle\eta(t)\rangle = 0$, $\langle\eta(t)\eta(t')\rangle = 2\gamma k_B T$.

- At the overdampped limit, $M\ddot{x} \approx 0$, so $-Kx - \gamma\dot{x} + \eta = 0$, and the mode is called relaxation mode with decay rate K/γ.
- If there is no water, $\gamma = 0$, so that the equation $M\ddot{x} = -Kx$ gives a vibrational mode with frequency $\sqrt{K/M}$.

The Langevin's equation for a general colloidal crystal or glass system consisting of n degrees of freedom can be written as

$$\mathbf{M}\ddot{\mathbf{x}} = -\mathbf{K}\mathbf{x} - \mathbf{\Gamma}\dot{\mathbf{x}} + \eta, \tag{3.2}$$

where $\mathbf{x}$ is the displacement of each particle from their equilibrium position, $\mathbf{K}$ describes the interaction of particles, e.g. charge force, $\mathbf{\Gamma}$ describes the hydrodynamics interaction, caused by motions in water. In general, the degrees of freedom can be translational or rational, so they have different M and γ, in this way $\mathbf{M}$ can be not written as $M/\gamma\mathbf{\Gamma}$, and $\mathbf{M}, \mathbf{K}, \mathbf{\Gamma}$ can not commute with each other. So even the equation is linear, the modes are coupled and there is no isolated modes shown in Eq. 3.1 in general (The isolated modes are composition of positions and velocities). However, in the following case, the modes are decoupled:

- At the overdampped limit, $\mathbf{M}\ddot{\mathbf{x}} \approx 0$, so the relaxation mode is determined by $\mathbf{K}$ and $\mathbf{\Gamma}$.
- In vacuum, the vibrational mode is determined by $\mathbf{M}$ and $\mathbf{K}$.

So in general, relaxiation modes are very different from the vibrational modes.

3.2 Numerical procedure

No matter in which case, one needs to obtain eigenvalues and eigenvectors of two symmetry matrices:

$$Ae = \lambda Be. \tag{3.3}$$

Since B are symmetric, so $B = RR^T$, and one obtains

$$R^{-1}AR^{-T}(R^Te) = \lambda(R^Te), \tag{3.4}$$

by solving the eigenvalues of $R^{-1}AR^{-T}$, one gets λ and the corresponding R^Te.

$\mathbf{K}$ can be obtain in terms of displacement covariance matrix $\mathbf{C} = \{\langle x_i x_j\rangle\}$:

$$\mathbf{C} = k_B T\mathbf{K}^{-1}. \tag{3.5}$$

3.3 Library routines

Remark: Even all files are .cpp, codes are in fact in pure C. Since the whole library calls Magick++ for image processing, which uses C++, so to simplify the Makefile and compilation, C++ suffix is used instead.

In the current library, only vibrational modes of ellipsoidal glasses are implemented.

The header files are in *include/normalmode.h*.

Core functions:

- *e_normalmode.cpp*: core function for calculation of normal modes.
- *e_DOS.cpp*: density of state (DOS)
- *e_DOS_log.cpp*: DOS in log scale, binned logarithmically
- *e_P.cpp*: participation of translational/rotational motions in modes
- *e_PR.cpp*: participation ratio
- *e_softspot.cpp*: identify soft spots
- *e_corr.cpp*: spatial correlations for modes

Plottings:

- *e_plot_config.cpp*: plot configurations of ellipsoidal glass
- *e_plot_mode.cpp*: plot modes with Gnuplot
- *e_plot_mode2.cpp*: plot modes with postscript
- *e_plot_mode3.cpp*: plot modes

- *e_plot_mode4.cpp*: plot modes

I/O:

- *e_dcm_io.cpp*: displacement correlation function I/O
- *e_ev_io.cpp*: eigenvalue and eigenvector I/O
- *e_mp_io.cpp*: mean position (approximately the equilibrium sites) I/O

Data process:

- *wrap_angle.cpp*: wrap the angle for periodic boundary conditions

Each function has a Unix-interface stored in *functions/*. Those functions can be served either as examples of using the library, or user interface in command line modes. Using of those functions can be found by running those commands without arguments.