Code Documentation

*Casey E. Berger**

January 10th, 2018

Contents

1 Code: "compile_data.py"

1. Read in the data from the HDF5 file, then compile all cfgs and make one .h5 file.

   Each file is a different configuration. Within each configuration, the data is in the following format:

   - ensemble (e.g. wf1p0_m51p3_l512_a52p0_smrw4p2_n60)
   - (e.g. mm)
   - quark mass (e.g. mlop0158_ms0p0902)
   - system (e.g. pipi, pik)
   - P_{CM} (e.g. PxoPyoPzo)
   - p_{rel} (e.g. pxopyopzo_qxoqyoqzo)
   - sources (e.g. xoy1oz8t11)
   - data: np array of shape (48,2,1) with complex data type
     48 - time steps
     2 - point and smeared sinks
     1 - this is a dummy index (can be different for other data, though - look out for it)

   `dmt.get_data(filename, system)` reads in this .h5 file and produces a dictionary, number of configurations, and a list of all the configuration numbers. The dictionary has format, e.g. for two pions: `pipi_dict["Pcm"]["prel"]["configuration"]["s`
   The entry with these keys will be an array of floats of shape (48,2) (the 2 is still the different sinks)

   The next function, `dmt.output_data(dict, output_filename, system)` outputs this dictionary to a new .h5 file for the system (currently only

*casey.e.berger@gmail.com

working with one ensemble and quark mass). This saves the new .h5, which has the same structure as the dictionary.

2. Average the sources together numerically (using numpy, not gvar)

   `dmt.get_compiled_data("filename")` reads in the compiled cfgs .h5 file and generates a dictionary, count of configurations, and list of configuration numbers. The dictionary has format: `pipi_dict["Pcm"]["prel"]["configuration"]["sou`

   Then, `dmt.average_sources(dict, system)` takes the dictionary and averages the different sources together for each P_CM, p_rel, and configuration and returns a new dictionary with format: `pipi_avg["Pcm"]["prel"]["configuration"`

   Finally, `dmt.output_averaged_sources(dict, output_filename, system, cfg_array = True/False)` saves the averaged data to a .h5 file with the same structure if cfg_array = False. If cfg_array = True, then it saves the .h5 file with format:

   - P_{CM} (e.g. PxoPyoPzo)
   - p_{rel} (e.g. pxopyopzo_qxoqyoqzo)
   - "Re"/"Im"
   - data: np array of shape (N_cfgs,48,2)
     48 - time steps
     2 - point and smeared sinks

3. Sum the shells together.

   `dmt.get_averaged_sources(dict, filename, system, cfg_array = True/False)` reads in a .h5 file and produces a dictionary, configuration count, and list of configurations. The dictionary has format: `pipi_avg["Pcm"]["prel"]["configuration"]["R`
   Next, `dmt.add_rel_momenta(dict, output_filename, system, cfg_array = True/False)` adds relative momenta in the same shell together with propoer weights for the pipi system and just reconfigures the dictionary for the single pion case (the shell for single pion is always "na"). In both cases, the dictionary is returned with format: `pipi_summed_shells["Pcm"]["shell"]["configu`
   "Shell" for the two pion case is equal to p^2_{rel} This is then saved (via `dmt.ouput_summed_shells(dict, output_filename, system)`) to an HDF5 file with format:

   - P_{CM} (e.g. PxoPyoPzo)
   - Shell (e.g. 0.0, na)
   - configuration
   - data: np array of shape (2,2,48)

4. Sum the P_{CM} together

   `dmt.read_in_summed_shells(dict, filename, system)` reads in a .h5 file and produces a dictionary and configuration count. The dictionary has format: `pipi_summed_shells["Pcm"]["shell"]["configuration"]["Re/Im"]`.
   Next, `dmt.average_CM(dict, output_filename, system` averages the center of mass momenta and returns a dictionary with format: `pipi_CM_averaged["Pcmsq"]["shel`

This is then saved (via `dmt.ouput_summed_shells(dict, output_filename, system)`) to an HDF5 file with format:

- P^2_{CM} (e.g. 0.0, 2.0, 4.0,...)
- Shell (e.g. 0.0)
- configuration
- Real/imaginary
- Point/smeared
- data: np array of shape (48,)

2 Code: "single_pion_fit.py"

You can adjust this to use just the smeared data (`nsinks = 1`) or both point and sink (`nsinks = 1`), and you can say how many states you want to fit (`nstates = 1,2,3` suggest starting with 1... right now it can go up to 3 but code can be adjusted to allow for more). Finally, you determine where to start the fit range, with `t_min = 14`. This code folds the data, so the fit range ends at t = 24 always.

1. Read in the data from the HDF5 file, using `read_in_CM_averaged(filename)` The file format is:

   - P_{CM} (e.g. PxoPyoPz0)
   - Shell (e.g. "Shell 0.0")
   - configuration
   - Real/imaginary
   - Point/smeared
   - data: np array of shape (48,)

   This returns a dictionary and the number of configurations. The dictionary has format: `data[pcmsq][shell][cfg][Re/Im][p/s]`

2. Format the data to be in the shape we want:

   It now looks like this: `dict["Pcm"]["shell"]["Re/Im"]["s/p"]` with the data in the shape np.array (N_{cfgs},t). This also returns the set of all configuration numbers in the data.

3. Set the fit range (using specifications from top of code)

4. Loop over the dictionary keys "Pcm", "shell", "Re/Im" and fold the s and p data.

5. (optional) plot the raw correlator, plus estimates for M_eff, overlap, etc. to help in choosing guesses for the fit coefficients

6. Cut the data to our fit range. Also concatenate them together if we are using both sinks.

7. Average the data together using gvar and generate the covariance matrix.
8. Fit the data, save the parameters, and print the ground state energy
9. Plot the fit
10. Now repeat this, but bootstrap. Generate a list of bootstrapped data points and then loop over them, feeding the bs data into the fitter and the plot function. (question - do I need to fit these, or do I only need to fit the averaged data at the end?) Also save the bootstrap data points to an array.
11. Average the list of bootstrap data points using gvar and then fit and plot.

3 Code: "pipi_fit.py"

Exactly the same as the single pion fit, but for two pions.