EISD Documentation

Release 0.1

David Brookes

April 11, 2016

CHAPTER

ONE

THIS IS DOCUMENTATION FOR EISD, AN IDP ENSEMBLE SCORING AND OPTIMIZATION PROGRAM FROM THE THG LAB AT UC BERKELEY.

CHAPTER TWO

TABLE OF CONTENTS

2.1 Getting Started

2.1.1 Introduction

Welcome to EISD! Short for Experimental Inferential Structure Determination, EISD was developed in Teresa Head-Gordon's lab at UC Berkeley with the goal of providing probability scores to ensembles of Intrinsically Disordered Proteins (IDPs) based on their fit to experimental data and conformational prior distributions. The theory underlying this code is presented in *[BrHe16]*.

EISD currently allows one to determine probabilities of ensembles using a number of different experimental data types and priors, as well as to optimize an ensemble by finding the subset of structures with the highest probability.

2.1.2 Dependencies

EISD relies on a number of Python libraries and external programs, listed below:

- **SciPy** *[JoOP01]*
- **Biopython** *[CACC09]*
- **SHIFTX2** *[BLGW11]*

2.1.3 Installation

To install EISD, one must first install the dependencies above. This is most easily done with the `pip` tool, which can be installed by following the instructions here. Then type on the command line:

```
$ pip install numpy
$ pip install sklearn
$ pip install biopython
```

Next, clone the git repository:

```
$ git clone https://github.com/davas301/EISD.git
```

Now navigate to the downloaded directory and run the setup tool:

```
$ cd eisd/
$ python setup.py
```

The EISD modules can now be imported into any local python program.

Users may also want to install **SHIFTX2** for chemical shift prediction, whose download and installation instructions can be found on their website

2.2 Use

2.2.1 Calculating EISD probabilities

The most basic use of EISD is to determine a probability score based on an ensemble of structure's fit to experimental data and some prior distribution on conformations. This requires back-calculating experimental observables from the ensemble's structure and determining the probability of the error between these values and the true experimental value. This procedure is currently implemented for J-coupling constants and chemical shifts. Prior probabilities are then calculated separately. EISD only currently implements a uniform prior distribution. See *[BrHe16]* for more details on these procedures.

Below is a tutorial on how to calculate these probabilities using EISD. This tutorial assumes that SHIFTX2 has been run on every structure in the ensemble and the output files are saved with the ".cs" extension

We begin by building the structures in the ensemble using `structure.Structure`:

```
import os
from eisd.structure import Structure

pdbdir = "$PATH_TO_ENSEMBLE_DIR$/"  # path to directory of PDB files

all_pdb_paths = [os.path.join(pdbdir, f) for f in os.listdir(pdbdir) if
                 ".pdb" in f and ".cs" not in f]

structs = [Structure(f, shiftxfile=f + ".cs") for f in all_pdb_paths]
```

Next we load in experimental data. The methods below (found in `readutil`) are specific to the data files in the `test` directory but can adapted for any experimental data format, as long as it returns a dictionary with `readutil.BaseDataID` objects as keys and `(observable, error)` tuples as values:

```
import readutil

jcoup_data = readutil.get_ab42_jcoup_data()
shift_data = readutil.get_ab42_shift_data()
```

Next we build our `backcalc.BaseBackCalculator` objects that will be used to approximately calculate experimental observables from structures information:

```
from eisd.backcalc import JCoupBackCalc, ShiftBackCalc

jcoup_back_calc = JCoupBackCalc()
shift_back_calc = ShiftBackCalc()
```

And now we use the above to build `DataEISD` objects that will calculate probabilities:

```
from eisd.eisd import DataEISD

shift_eisd = DataEISD(shift_back_calc, shift_data)

# don't consider back-calc error for j-coupling (error is all in nuisance parameter variability):
jcoup_eisd = DataEISD(jcoup_back_calc, jcoup_data, no_bc_err=True)
```

Now build a *priors.UniformPrior* object assuming there are 10 ensembles in our hypothesis space (so the prior probability of each is 0.1):

```
from eisd.priors import UniformPrior

prior = UniformPrior(10)
```

Now we calculate and print the probability of the ensemble given the experimental data and prior distribution:

```
shift_logp = shift_eisd.calc_logp(structs)
jcoup_logp = jcoup_eisd.calc_logp(structs)
prior_logp = prior.calc_prior_logp()
total_logp = shift_logp + jcoup_logp + prior_logp()

outstr = "The ensemble at %s fits chemical shift data with log probability of %.3f, " \
         "J coupling data with log probability of %.3f and a conformational prior" \
          "distribution with log probability %.3f" The total log probability is then %.3f" \
          % (pdbdir, shift_logp, jcoup_logp, prior_logp, total_logp)
print outstr
```

2.2.2 Finding optimal subsets

Due to EISD's size extensivity, it is ideal for finding the optimal subset of an ensemble that maximizes the probabilities found above. This optimization is implemented in the using a simulated annealing to swap out structures in the subset with structures from the full ensemble.

Below is an example showing how to perform this optimization with J-coupling and chemical shift experimental data and a Uniform Prior. Again, this assumes that SHIFTX2 has been run for every structure in the full ensemble.

First define the directory where the full ensemble of structures is stored. This is referred to as the "reservoir" of structures. Also define the size of the subset that will be optimized and the number of iterations of simulated annealing to be performed. Additionally provide the path to a file that will contain the names of the pdb files that make up the optimal ensemble, as well as a file for storing optimization statistics:

```
pdbdir = "$PATH_TO_ENSEMBLE_DIR$/"
subset_size = 1000
niter = int(1e6)

savefile = "../output/$OPT_FILE$"
stats_file = "../output/$STATS_FILE$"
```

Now build a *priors.BasePrior* object and *eisd.DataEISD* objects for each set of experimental data:

```
from eisd.priors import UniformPrior
from eisd.eisd import DataEISD
from backcalc import JCoupBackCalc, ShiftBackCalc
import eisd.readutil

prior = UniformPrior(niter)  # assume every iterations produces a new hypothesis ensemble

jcoup_eisd = DataEISD(JCoupBackCalc(), eisd.readutil.get_ab42_jcoup_data(),
                      no_bc_err=True)
shift_eisd = DataEISD(ShiftBackCalc(), eisd.readutil.get_ab42_shift_data())

# put eisd's together in a list:
data_eisds = [jcoup_eisd, shift_eisd]
```

Now define a cooling schedule for simulated annealing. Below is the default schedule:

```
def cool_sched(t):
    """
    Cooling schedule

    :param t: fraction of iterations
    :return: "Temperature" for simulated annealing
    """
    return np.exp(-(2 * t) ** 2)
```

Now we can build the `eisd.EISDOPT` object and begin the optimization:

```
from eisd.eisd import EISDOPT

optimizer = EISDOPT(pdbdir, prior, data_eisds, savefile,
                    subsize=subset_size, verbose=True,
                    stats_file=stats_file)

optimizer.opt(niter, cool_sched=cool_sched)
```

2.3 Modules

2.3.1 eisd

class `eisd.`**`DataEISD`**(*back_calc*, *exp_data*, *no_exp_err=False*, *no_params=False*, *no_bc_err=False*, *no_opt=False*)

Implementation of the Experimental Inferential Structure Determination model. This does not include prior probabilities (priors should be calculated separately). Each DataEISD object calculates probabilities for one data type

Parameters

- **`back_calc`** – BaseBackCalculator object
- **`exp_data`** – experimental data corresponding to back-calc
- **`no_exp_err`** – True if experimental error should not contribute
- **`no_params`** – True if parameter probabilities should not contribute
- **`no_bc_err`** – True if back-calculation error should not contribute
- **`no_opt`** – True if no optimization should be performed–just calculate error probability from normal

`_eval`(*structs*, *params*, *bc_err*, *j*)

Given a list of structures, bc_err, params and a data point index, return the experimental error at the data point. This essentially implements equation 5 in Brookes (2016).

Parameters

- **`structs`** – list of Structure objects
- **`params`** – back-calc parameters
- **`bc_err`** – back-calculation error
- **`j`** – index of data point

Returns experimental error

`_logp_bc_err`(*value*, *j*)
Log probability of back-calculation error

Parameters

- **`value`** – value of error
- **`j`** – index of data point

Returns log probability

`_logp_exp_err`(*value*, *j*)
Log probability of experimental error

Parameters

- **`value`** – error value
- **`j`** – index of experimental data point

Returns log probability

`_logp_params`(*value*)
Calculate the log probability of back-calculator parameters

Parameters **`value`** – back-calculator parameters

Returns log probability of parameters

`_logp_total_err`(*value*, *j*)
For the no-opt sceneario. Return log probability of an error at data point j assuming the error distribution is a normal with variance equal to the sum of the experimental and back-calculator error variances

Parameters

- **`value`** – value of error
- **`j`** – index of data point

Returns log probability

`_random_bc_err`(*j*)
Return a random back-calculator error

Parameters **`j`** – index of data point

Returns random error

`_random_exp_err`(*j*)
Return a random experimental error

Parameters **`j`** – index of experimental data point

Returns random error

`_random_params`()
See BaseBackCalculator for more info

Returns random parameter set

`calc_logp`(*structs*)
Calculate the full EISD log probability given an ensemble :param structs: list of Structure objects

Returns log probability

`compute_all_back_calc`(*structs*, *bc_params*, *j*)
Perform back-calculation on every input structure for data point j

Parameters

- **`structs`** – list of Structure objects
- **`bc_params`** – input parameters for back-calculator
- **`j`** – index of data point to back-calculate for

Returns vector of back-calculations

`compute_back_calc_mean`(*structs*, *bc_params*, *j*)
Compute mean of back-calculation on input structures for data point j

Parameters

- **`structs`** – list of Structure objects
- **`bc_params`** – input parameters for back-calculator
- **`j`** – index of data point to back-calculate for

Returns mean of back-calculations

class `eisd.`**`EISDOPT`**(*pdb_dir*, *prior*, *data_eisds*, *savefile*, *subsize=1000*, *verbose=True*, *stats_file=None*)
Class for optimizing an ensemble based on EISD probabilities

Parameters

- **`pdb_dir`** – path to full ensemble of pdb structures
- **`prior`** – a BasePrior object
- **`data_eisds`** – list of DataEISD objects
- **`savefile`** – file to save best ensemble to (list of pdb names)
- **`subsize`** – size of sub-ensemble to optimize
- **`verbose`** – if True, will print updates
- **`stats_file`** – file to save statistics to (optional)

`_build_start_set`()
Build a random start set of structures

Returns list of files and list of Structure objects

`_calc_data_prob`()
Calculate the probability of the data for the current set of structures

Returns probability of data

`_perturb`()
Randomly remove a structure and add a random structure from the reservoir

`_restore`()
Restore the system to its state before the last perturbation

static **`default_cool`**(*t*)
Default cooling schedule for simulated annealing

Parameters **`t`** – fraction of iterations

Returns "Temperature" for simulated annealing

opt (*niter*, *cool_sched=None*)
Perform simulated annealing procedure to find the subset of structures that maximimizes probability. Requires the number of iterations and the cooling schedule, which is a callable function that takes the current fraction of iterations performed

Parameters

- **niter** – number of iterations
- **cool_sched** – callable cooling schedule function

2.3.2 backcalc

class backcalc.**BaseBackCalculator** (*nparams*)
Abstract base class for back calculators

Parameters **nparams** – number of nuisance parameters

back_calc (*xi*, *params*)
Perform the back-calculation given a structural measurement and necessary parameters.

Parameters

- **xi** – a Measurement returned by self.get_struct_data()
- **params** – list of nuisance parameter values for this back-calc

Returns a Measurement object containing the back-calculated value

get_default_err (*data_id*)
Get the default (i.e. most probable) error for this back-calculator

Parameters **data_id** – a DataID object

Returns the default error corresponding to the input data id

get_default_params ()
Get the default parameters for this back-calculator

Returns List of default values (one for each param)

get_default_struct_val ()
Returns a default structural value for the relevant structural information needed by this back-calculator. Used to initialize models

Returns default Measurement object

get_err_sig (*data_id*)
Get error standard deviation for a data point

Parameters **data_id** – DataID object for the data point

get_random_err (*data_id*)
Get a value drawn from the error distribution of this back-calculator

Parameters **data_id** – a DataID object

Returns a random error taken from the distribution corresponding to the input data id

get_random_params ()
Get values drawn randomly from the distributions of this back-calculator's parameters

Returns List of random values (one for each param)

logp_err(*err*, *data_id*)
Return the log probability of an error

Parameters

- **err** – the error value
- **data_id** – the DataID corresponding to the input error

Returns the log probability of the input error

logp_params(*params*)
Return the log probability of a set of parameter values

Parameters **params** – list of input parameters for this back-calculator

Returns log probability of the input parameters

class backcalc.**JCoupBackCalc**(*err_mu=0*, *err_sig=0.73*, *mu_a=6.51*, *mu_b=-1.76*, *mu_c=1.6*, *sig_a=0.37416573867739417*, *sig_b=0.17320508075688773*, *sig_c=0.28284271247461901*)
Back calculator for J-coupling constants from dihedral angles Calculation is done with the Karplus equation: $$ J(phi) = Acos^2(phi) + Bcos(phi) + C $$ Constructor requires parameters for the Gaussian random variables that will represent the coefficients in the Karplus equation and a dictionary containing the errors mean and standard deviations for every type of structural measurement. Default Karplus coeff means, stdevs and and error stdev are the values based on those in:

Vuister and Bax, "Quantatative J Correlation: a new approach for meausuring homonuclear three bond J coupling constants in N15 eriched proteins" *J.Am.Chem.Soc*, **1993**, *115* (17). pp 7772:7777

back_calc(*xi*, *back_params*)
Implemenation of the Karplus equation. See BaseBackCalculator for more info

Parameters

- **xi** –
- **back_params** –

get_default_err(*data_id=None*)
Default if mean of error distribution See BaseBackCalculator for more info

Parameters **data_id** –

Returns

get_default_params()
Return mean of param values. See BaseBackCalculator for more info

Returns

get_default_struct_val()
Default is [pi, pi] dihedral angle. See BaseBackCalculator for more info

Returns

get_err_sig(*data_id=None*)
See BaseBackCalculator for more info

Parameters **data_id** –

Returns

get_random_err(*data_id=None*)
data_id is optional for this back-calculator. See BaseBackCalculator for more info

Parameters **data_id** –

get_random_params()
See BaseBackCalculator for more info

Returns

logp_err(*err*, *data_id=None*)
Error is represemted as a normal. See BaseBackCalculator for more info

Parameters

- **err** –
- **data_id** –

logp_params(*params*)
Parameters are represented as normals. See BaseBackCalculator for more info

Parameters **params** –

Returns

class backcalc.**ShiftBackCalc**
Back calculator for converting pdb files to chemical shifts. Uses SHIFTX2 for the back-calculation, which has no nuisance parameters

back_calc(*xi*, *params*)
In this case, xi contains the back-calculated measurement. See BaseBackCalculator for more info.

Parameters

- **xi** –
- **params** –

Returns

get_default_err(*data_id=None*)
Default is 0.0. See BaseBackCalculator for more info

Parameters **data_id** –

Returns

get_default_params()
No nuisance parameters required for this model. See BaseBackCalculator for more info

Returns

get_default_struct_val()
Return a shift id for the first measurement of the first structure

Returns

get_err_sig(*data_id*)
Get the std corresponding to the SHIFTX2 rmsd for this data id

Parameters **data_id** –

Returns

get_random_err(*data_id*)
Return a random value from the normal distribution corresponding to the SHIFTX2 rmsd value for this data id. See BaseBackCalculator for more info

Parameters **data_id** –

Returns

get_random_params()

No nuisance parameters required for this model. See BaseBackCalculator for more info

Returns

logp_err(*err*, *data_id*)

Error represented as gaussian with std based on SHIFTX2 reported rmsd See BaseBackCalculator for more info

Parameters

- **err** –
- **data_id** –

Returns

logp_params(*params=None*)

No nuisance parameters required for this model. See BaseBackCalculator for more info

Parameters **params** –

Returns

2.3.3 priors

class priors.**BasePrior**

Abstract base class for prior distributions

calc_prior_logp(**args*)

Calculate the prior log probability for a list of Structures that make up an ensemble

Parameters **args** – specific arguments

Returns log probability of ensemble in this prior distribution

get_arg(*struct*)

Build the arguments required to calculate the prior given a single structure. So the input to calc_prior_logp shoudl be a list of these args

Parameters **struct** – a Structure object

Returns

class priors.**UniformPrior**(*n*)

Uniform prior distribuiton across space of ensembles

Parameters **n** – number of candidate ensembles

calc_prior_logp()

Probability is just 1/n. See BasePrior for more info

Returns

get_arg(*struct=None*)

No arguments for this prior. See BasePrior for more info

Parameters **struct** – a Structure object

Returns None

2.3.4 structure

class `structure.`**`Structure`**(*pdbfile*, *shiftxfile=None*, *runshiftx=None*, *energy=None*)

Class for representing and modifying protein structures. Interfaces significantly with MMTK.

Parameters

- **`pdbfile`** – path to the pdbfile containing the representation of this structure
- **`shiftxfile`** – file containing back-calculated SHIFTX2 data for this structure
- **`runshiftx`** – an optional RunShiftX instance

`_get_all_dihed`()

Retrieve all phi, psi dihedral angles in this structure

Returns a {res_num: (phi, psi)} dict

`get_struct_measure`(*exp_id*)

Get a structural measurement from a DataID corresponding to an experimental measurement. This structural measurement can be input into a back-calculator

Parameters **`exp_id`** – An experimental data ID (ShiftID or JCoupID)

2.3.5 readutil

class `readutil.`**`BaseDataID`**

Base class for objects that contain unique IDs for data points. The goal of these classes is to ensure that experimental measurements are solidly connected to the corresponding structural and back-calculation measurements. These IDs must be hashable and have equality comparisons

class `readutil.`**`JCoupID`**(*res_num*)

ID objects for j coupling values. Only requires a residue number

Parameters **`res_num`** – residue number of measurement

class `readutil.`**`Measurement`**(*data_id=None*, *val=None*)

Class for storing all of the info in a data measurement (experimental or structural). Contains a data ID and a value

Parameters

- **`data_id`** – a BaseDataID object
- **`val`** – value of the measurement

class `readutil.`**`RunShiftX`**(*exe='/usr/local/bin/shiftx2/shiftx2.py'*, *tempdir='./tmp/'*)

Class for running the SHIFTX2 command line program

Parameters

- **`exe`** – path to SHIFTX2 executable
- **`tempdir`** – path to a directory to story temporary files

`_clean`()

Removes temporary files

static **`read_ouput`**(*f*)

Reads a shiftx2 output file

Parameters **`f`** – output file to be read

Returns {ShiftID: shift_val} dict

run_shiftx_once(*inpath*, *backbone_only=False*, *clean=True*, *no_output=True*)
Runs SHIFTX2 for a single input pdb file

Parameters

- **inpath** – path to input pdb file
- **backbone_only** – True if SHIFTX2 should only run for backbone atoms
- **clean** – True if the temporary files should be deleted
- **no_output** – If true, no SHIFTX2 ouput will be shown on

Returns path to output ".cs" file

class readutil.**ShiftID**(*res_num*, *atom_name*)
Storage objects containing information that identifies a unique chemical shift. A shift is defined by a residue number and atom name

Parameters

- **res_num** – residue number of measurement
- **atom_name** – name of atom with residue

readutil.**get_ab42_jcoup_data**()
Read the J-coupling data for aB42 in ../test_data/

Returns a {JCoupID: val} dict

readutil.**get_ab42_shift_data**()
Read the shift data for aB42 in ../test_data/

Returns a {ShiftID: val} dict

readutil.**get_md_energies**()
Read the energy file containing all the energies for the MD ensemble in in ../test_data/

Returns {filename: energy} dict

readutil.**read_dihed_file**(*path*)
Read the dihedral angle file written by write_dihed_to_file

Parameters **path** –

Returns numpy array containg dihedral angles

readutil.**write_dihed_to_file**(*structs*, *outname*, *verbose=True*)
Write the dihedral angles of a list of structures into a tab-separated file where each line represents a single structure and the columns alternate phi, psi angles for each residue

Parameters

- **structs** – list of Structure objects
- **outname** – path to where the dihed file should be written
- **verbose** – If True updates will be written to terminal

2.3.6 util

util.**normal_loglike**(*x*, *mu*, *sig*)
Returns the log likelihood of a value x in a normal distribution with mean mu and stdev sig.

Parameters

- **`x`** – list of values to calculate probability of
- **`mu`** – mean of distribuiton
- **`sig`** – standard deviation of distribution

2.4 References

References

2.5 License

This program is distributed under the BSD license:

[BrHe16] Brookes, David H. and Teresa Head-Gordon. “Experimental Inferential Structure Determination of Ensembles for Intrinsically Disordered Proteins”. *J. Am. Chem. Soc.*, **2016**, *138* (13), pp 4530–4538

[JoOP01] Jones E, Oliphant E, Peterson P, *et al*. SciPy: Open Source Scientific Tools for Python, 2001-.

[CACC09] Peter J.A Cock, et al. “Biopython: freely available Python tools for computational molecular biology and bioinformatics”. *Bioinformatics*, **2009**, *25* (11), pp 1422-1423.

[BLGW11] Beomsoo Han, Yifeng Liu, Simon Ginzinger, and David Wishart. “SHIFTX2: significantly improved protein chemical shift prediction”. *Journal of Biomolecular NMR*, **2011**, 50 (1), pp 43-57.

PYTHON MODULE INDEX