MELD Documentation

Release 1.0

Justin MacCallum

August 16, 2014

Contents

In the works of Stone, a predominant concept is the concept of modernist art. Thus, the subject is contextualised into a pretextual objectivism that includes culture as a whole.

Baudrillard promotes the use of neocultural narrative to analyse language. But a number of theories concerning not, in fact, situationism, but subsituationism may be discovered.

The premise of subtextual construction suggests that reality is capable of deconstruction. Thus, an abundance of theories concerning socialist realism exist.

Look how easy it is to use:

```
import project
# Get your stuff done
project.do_stuff()
```

CHAPTER 1

Contents

1.1 User Guide

Release 1.0

Date August 16, 2014

1.1.1 Article 1

The main theme of the works of Stone is the role of the artist as observer. The primary theme of Dahmus's[1] model of neocultural narrative is a mythopoetical reality. However, socialist realism implies that truth serves to disempower the underprivileged.

In the works of Burroughs, a predominant concept is the distinction between closing and opening. Marx suggests the use of semantic discourse to attack the status quo. Therefore, if socialist realism holds, we have to choose between subtextual construction and the postcultural paradigm of context.

"Sexual identity is fundamentally used in the service of sexism," says Baudrillard; however, according to Parry[2] , it is not so much sexual identity that is fundamentally used in the service of sexism, but rather the rubicon of sexual identity. The subject is interpolated into a neocultural narrative that includes sexuality as a paradox. It could be said that Derrida promotes the use of modern subdeconstructivist theory to deconstruct and read class.

If one examines subtextual construction, one is faced with a choice: either reject neocultural narrative or conclude that government is capable of significance, but only if the premise of subtextual construction is valid; if that is not the case, we can assume that reality is used to entrench outdated perceptions of society. Foucault uses the term 'socialist realism' to denote not narrative, as Sartre would have it, but neonarrative. But Sontag suggests the use of dialectic discourse to attack sexism.

Finnis[3] states that we have to choose between neocultural narrative and precapitalist theory. Therefore, socialist realism holds that class has intrinsic meaning.

The subject is contextualised into a neocultural narrative that includes sexuality as a reality. But Bataille promotes the use of subtextual construction to challenge society.

If Baudrillardist simulacra holds, we have to choose between socialist realism and conceptualist socialism. However, the subject is interpolated into a neocultural narrative that includes art as a totality.

Any number of discourses concerning a self-falsifying paradox may be revealed. Therefore, the premise of subtextual construction implies that culture is dead.

The subject is contextualised into a subcultural paradigm of reality that includes truth as a reality. However, in Foucault's Pendulum, Eco deconstructs neocultural narrative; in The Name of the Rose, although, he analyses Debordist image.

The characteristic theme of the works of Eco is not theory, but posttheory. Thus, von Junz[4] suggests that we have to choose between socialist realism and precultural theory.

1.1.2 Article 2

In the works of Eco, a predominant concept is the concept of subdialectic reality. An abundance of appropriations concerning capitalist feminism exist. However, if socialist realism holds, we have to choose between the pretextual paradigm of context and dialectic narrative.

The main theme of Hanfkopf's[5] critique of textual deconstruction is the difference between society and culture. The subject is interpolated into a socialist realism that includes truth as a whole. Thus, Parry[6] implies that we have to choose between subtextual construction and cultural rationalism.

The primary theme of the works of Eco is the paradigm, and some would say the dialectic, of postcapitalist class. However, the example of Marxist class prevalent in Eco's The Aesthetics of Thomas Aquinas emerges again in The Island of the Day Before.

Sontag's model of textual deconstruction states that the goal of the poet is social comment, given that culture is distinct from narrativity. Therefore, Marx suggests the use of socialist realism to deconstruct sexist perceptions of culture.

The subject is contextualised into a textual deconstruction that includes language as a totality. In a sense, if socialist realism holds, we have to choose between textual deconstruction and dialectic capitalism.

1.1.3 Article 3

In the works of Eco, a predominant concept is the distinction between ground and figure. The characteristic theme of Porter's[7] essay on textual deconstruction is the role of the reader as artist. It could be said that Baudrillard promotes the use of socialist realism to read and modify sexual identity.

"Society is intrinsically impossible," says Lyotard. Foucault uses the term 'textual deconstruction' to denote the stasis, and eventually the collapse, of constructivist consciousness. Therefore, the primary theme of the works of Fellini is the common ground between society and class.

If one examines subtextual construction, one is faced with a choice: either accept socialist realism or conclude that expression is a product of communication. Subdialectic desublimation implies that the raison d'etre of the observer is significant form. Thus, the main theme of Prinn's[8] critique of subtextual construction is the role of the reader as poet.

Several narratives concerning a pretextual reality may be discovered. Therefore, the subject is interpolated into a socialist realism that includes reality as a paradox.

Sartre uses the term 'dialectic discourse' to denote the dialectic, and some would say the collapse, of subcapitalist narrativity. It could be said that the characteristic theme of the works of Fellini is the role of the reader as observer.

The subject is contextualised into a socialist realism that includes truth as a whole. But the primary theme of McElwaine's[9] essay on textual deconstruction is not desituationism as such, but neodesituationism.

Debord suggests the use of socialist realism to challenge sexism. However, Humphrey[10] states that we have to choose between subtextual construction and semioticist predialectic theory.

1.2 API Reference

Release 1.0

Date August 16, 2014

1.2.1 meld.comm

class meld.comm.**MPICommunicator**(*n_atoms*, *n_replicas*)
Class to handle communications between master and slaves using MPI.

Parameters

- **n_atoms** – number of atoms
- **n_replicas** – number of replicas

Note: creating an MPI communicator will not actually initialize MPI. To do that, call `initialize()`.

---

broadcast_alphas_to_slaves(*alphas*)
Send the alpha values to the slaves.

Parameters **alphas** – a list of alpha values, one for each replica. The master node's alpha value should be included in this list. The master node will always be at alpha=0.0

Returns `None`

broadcast_states_for_energy_calc_to_slaves(*states*)
Broadcast states to all slaves. Send all results from this step to every slave so that we can calculate the energies and do replica exchange.

Parameters **states** – a list of states

Returns `None`

broadcast_states_to_slaves(*states*)
Send a state to each slave.

Parameters **states** – a list of states. The list of states should include the state for the master node. These are the states that will be simulated on each replica for each step.

Returns the state to run on the master node

exchange_states_for_energy_calc(*state*)
Exchange states between all processes.

Parameters **state** – the state for this node

Returns a list of states from all nodes

gather_energies_from_slaves(*energies_on_master*)
Receive a list of energies from each slave.

Parameters **energies_on_master** – a list of energies from the master

Returns a square matrix of every state on every replica to be used for replica exchange

gather_states_from_slaves(*state_on_master*)
Receive states from all slaves

Parameters **state_on_master** – the state on the master after simulating

Returns A list of states, one from each replica. The returned states are the states after simulating.

initialize()
Initialize and start MPI

is_master()
Is this the master node?

Returns `True` if we are the master, otherwise `False`

receive_alpha_from_master()
Receive alpha value from master node.

Returns a floating point value for alpha in [0,1]

receive_state_from_master()
Get state to run for this step

Returns the state to run for this step

receive_states_for_energy_calc_from_master()
Receive all states from master.

Returns a list of states to calculate the energy of

send_energies_to_master(*energies*)
Send a list of energies to the master.

Parameters **energies** – a list of energies to send to the master

Returns None

send_state_to_master(*state*)
Send state to master

Parameters **state** – State to send to master. This is the state after simulating this step.

Returns None

1.2.2 meld.parse

class meld.parse.**SecondaryRun**
SecondaryRun(start, end)

end
Alias for field number 1

start
Alias for field number 0

meld.parse.**get_secondary_structure_restraints**(*system*, *scaler*, *torsion_force_constant=2.48*, *distance_force_constant=2.48*, *quadratic_cut=2.0*, *filename=None*, *contents=None*, *file=None*)
Get a list of secondary structure restraints.

Parameters

- **system** – a System object
- **scaler** – a force scaler
- **torsion_force_constant** – force constant for torsions, in kJ/mol/(10 degree)^2
- **distance_force_constant** – force constant for distances, in kJ/mol/Angstrom^2
- **quadratic_cut** – switch from quadratic to linear beyond this distance, Angstrom
- **filename** – string of filename to open
- **contents** – string of contents to process
- **file** – file-like object to read from

Returns a list of RestraintGroups

Note: specify exactly one of filename, contents, file.

meld.parse.**get_sequence_from_AA1**(*filename=None*, *contents=None*, *file=None*)
Get the sequence from a list of 1-letter amino acid codes.

Parameters

- **filename** – string of filename to open
- **contents** – string containing contents
- **file** – a file-like object to read from

Returns a string that can be used to initialize a system

Raise RuntimeError on bad input

Note: specify exactly one of filename, contents, file

meld.parse.**get_sequence_from_AA3**(*filename=None*, *contents=None*, *file=None*)
Get the sequence from a list of 3-letter amino acid codes.

Parameters

- **filename** – string of filename to open
- **contents** – string containing contents
- **file** – a file-like object to read from

Returns a string that can be used to initialize a system

Raise RuntimeError on bad input

Note: specify exactly one of filename, contents, file

1.2.3 meld.pdb_writer

1.2.4 meld.remd.adaptor

class meld.remd.adaptor.**AcceptanceCounter**(*n_replicas*)
Class to keep track of acceptance rates.

class meld.remd.adaptor.**AdaptationPolicy**(*growth_factor*, *burn_in*, *adapt_every*)
Repeat adaptation on a regular schedule with an optional burn-in and increasing adaptation times.

Parameters

- **growth_factor** – increase adapt_every by a factor of growth_factor every adaptation
- **burn_in** – number of steps to ignore at the beginning
- **adapt_every** – how frequently to adapt (in picoseconds)

class AdaptationRequired
AdaptationRequired(adapt_now, reset_now)

adapt_now
Alias for field number 0

reset_now
Alias for field number 1

AdaptationPolicy.**should_adapt**(*step*)
Is adaptation required?

Parameters **step** – the current simulation step

Returns an AdaptationPolicy.AdaptationRequired object indicating if adaptation or resetting is necessary

class meld.remd.adaptor.**EqualAcceptanceAdaptor**(*n_replicas*, *adaptation_policy*, *min_acc_prob=0.1*)
Adaptor based on making acceptance rates uniform.

Parameters

- **n_replicas** – number of replicas
- **min_acc_prob** – all acceptence probabilities below this value will be raised to this value

adapt(*previous_lambdas*, *step*)
Compute new optimal values of lambda.

Parameters

- **previous_lambdas** – a list of the previous lambda values
- **step** – the current simulation step

Returns a list of the new, optimized lambda values

reset()
Forget about any previous updates.

Resets all internal counters and statistics to zero.

update(*i*, *accepted*)
Update adaptor with exchange.

Parameters

- **i** – index of first replica; second replica is i+1
- **accepted** – True if the exchange was accepted

1.2.5 meld.remd.ladder

class meld.remd.ladder.**NearestNeighborLadder**(*n_trials*)
Class to compute replica exchange swaps between neighboring replicas.

Parameters **n_trials** – total number of replica exchange swaps to attempt

compute_exchanges(*energies*, *adaptor*)
Compute the exchanges given an energy matrix.

Parameters

- **energies** – numpy array of energies (see below for details)
- **adaptor** – replica exchange adaptor that is updated every attempted swap

Returns a permutation vector (see below for details)

The energy matrix should be an n_rep x n_rep numpy array. All energies are in dimensionless units (unit of kT). Each column represents a particular structure, while each row represents a particular combination of temperature and Hamiltonian. So, `energies[i,j]` is the energy of structure j with temperature and hamiltonian i. The diagonal `energies[i,i]` are the energies that were actually simulated.

This method will attempt `self.n_trials` swaps between randomly chosen pairs of adjacent replicas. It will return a permutation vector that describes which index each structure should be at after swapping. So, if the output was `[2, 0, 1]`, it would mean that replica 0 should now be at index 2, replica 1 should now be at index 0, and replica 2 should now be at index 1. Output of `[0, 1, 2]` would mean no change.

The adaptor object is called once for each attempted exchange with the indices of the attempted swap and the success or failure of the swap.

1.2.6 meld.remd.launch

1.2.7 meld.remd.master_runner

class meld.remd.master_runner.**MasterReplicaExchangeRunner**(*n_replicas*, *max_steps*, *ladder*, *adaptor*, *ramp_steps=None*)

Class to coordinate running of replica exchange

This class doesn't really know much about the calculation that is happening, but it's the glue that holds everything together.

Parameters

- **n_replicas** – number of replicas
- **max_steps** – maximum number of steps to run
- **ladder** – Ladder object to handle exchanges
- **adaptor** – Adaptor object to handle alphas adaptation
- **ramp_steps** – integer number of steps to ramp up force constants at start of simulation

run(*communicator*, *system_runner*, *store*)

Run replica exchange until finished

Parameters

- **communicator** – A communicator object to talk with slaves
- **system_runner** – a ReplicaRunner object to run the simulations
- **store** – a Store object to handle storing data to disk

to_slave()

Return a SlaveReplicaExchangeRunner based on self.

1.2.8 meld.remd.multiplex_runner

class meld.remd.multiplex_runner.**MultiplexReplicaExchangeRunner**(*n_replicas*, *max_steps*, *ladder*, *adaptor*, *ramp_steps=None*)

Class to coordinate running of replica exchange

Parameters

- **n_replicas** – number of replicas
- **max_steps** – maximum number of steps to run
- **ladder** – Ladder object to handle exchanges

- **adaptor** – Adaptor object to handle alphas adaptation
- **ramp_steps** – integer number of steps to ramp up force constants at start of simulation

run (*system_runner*, *store*)
Run replica exchange until finished

Parameters

- **system_runner** – a ReplicaRunner object to run the simulations
- **store** – a Store object to handle storing data to disk

1.2.9 meld.remd.reseed

class meld.remd.reseed.**NullReseeder**
Dummy reseeder that does nothing.

class meld.remd.reseed.**Reseeder** (*interval*, *candidate_frames*)
Reseed replicas from previous states.

This class implements a reseeder that will periodically reseed all replicas with structures from the past history of the lowest replica.

Parameters

- **interval** – number of timesteps between reseeding
- **candidate_frames** – consider this many frames previous to the current frame for reseeding

reseed (*step*, *current_states*, *store*)
Perform the reseeding.

Parameters

- **step** – the current timestep
- **current_states** – a list of the current replica states to be modified
- **store** – a DataStore object to get historical structures from

1.2.10 meld.remd.slave_runner

class meld.remd.slave_runner.**SlaveReplicaExchangeRunner** (*step*, *max_steps*, *ramp_steps=None*)
This class coordinates running replica exchange on the slaves.

classmethod from_master (*master*)
Initialize a new slave from a master.

Parameters **master** – a meld.remd.master_runner.MasterReplicaExchangeRunner to serve as a template

Returns a SlaveReplicaExchangeRunner

run (*communicator*, *system_runner*)
Continue running slave jobs until done.

Parameters

- **communicator** – a communicator object for talking to the master
- **system_runner** – a system_runner object for actually running the simulations

1.2.11 meld.system.builder

1.2.12 meld.system.openmm_runner.cmap

class meld.system.openmm_runner.cmap.**CMAPResidue**
CMAPResidue(res_num, res_name, index_N, index_CA, index_C)

index_C
Alias for field number 4

index_CA
Alias for field number 3

index_N
Alias for field number 2

res_name
Alias for field number 1

res_num
Alias for field number 0

1.2.13 meld.system.openmm_runner.runner

1.2.14 meld.system.openmm_runner.softcore

1.2.15 meld.system.protein

class meld.system.protein.**ProteinBase**
Base class for other Protein classes.

Provides functionality for translation/rotation and adding H-bonds.

add_disulfide(*res_index_i*, *res_index_j*)
Add a disulfide bond.

Parameters

- **res_index_i** – one-based index of residue i
- **res_index_j** – one-based index of residue j

Note: indexing starts from one and the residue numbering from the PDB file is ignored. When loading from a PDB or creating a sequence, residue name must be CYX, not CYS.

---

set_rotation(*rotation_axis*, *theta*)
Set the rotation.

Parameters

- **rotation_axis** – `numpy.array(3)` in nanometers
- **theta** – angle of rotation in degrees

Rotation happens after translation.

set_translation(*translation_vector*)
Set the translation vector.

Parameters **translation_vector** – `numpy.array(3)` in nanometers

Translation happens after rotation.

class meld.system.protein.**ProteinMoleculeFromPdbFile**(*pdb_path*)

Create a new protein molecule from a pdb file. This class is dumb and relies on AmberTools for the heavy lifting.

Parameters **pdb_path** – string path to the pdb file

Note: no processing happens to this pdb file. It must be understandable by tleap and atoms/residues may need to be added/deleted/renamed. These manipulations should happen to the file before MELD is invoked.

---

class meld.system.protein.**ProteinMoleculeFromSequence**(*sequence*)

Class to create a protein from sequence. This class will create a protein molecule from sequence. This class is pretty dumb and relies on AmberTools to do all of the heavy lifting.

Parameters **sequence** – sequence of the protein to create

The sequence is specified in Amber/Leap format. There are special NRES and CRES variants for the N- and C-termini. Different protonation states are also available via different residue names. E.g. ASH for neutral ASP.

1.2.16 meld.system.restraints

class meld.system.restraints.**AlwaysActiveCollection**

class meld.system.restraints.**ConfinementRestraint**(*system*, *scaler*, *res_index*, *atom_name*, *radius*, *force_const*)

Confinement restraint

Parameters

- **system** – a System object
- **scaler** – a force scaler
- **res_index** – integer, starting from 1
- **atom_name** – atom name
- **raidus** – calculed couplings within tolerance (in Hz) of d_obs will have zero energy and force
- **force_const** – force sonstant in $kJ/mol/Hz^2$

Confines an atom to be within radius of the origin. These restraints are typically set to somewhat larger than the expected radius of gyration of the protein and help to keep the structures comapct even when the protein is unfolded. Typically used with a ConstantScaler.

class meld.system.restraints.**ConstantScaler**

This scaler is "always on" and always returns a value of 1.0".

class meld.system.restraints.**DistanceRestraint**(*system*, *scaler*, *atom_1_res_index*, *atom_1_name*, *atom_2_res_index*, *atom_2_name*, *r1*, *r2*, *r3*, *r4*, *k*)

Distance restraint

Parameters

- **system** – a System object
- **scaler** – a force scaler
- **atom_1_res_index** – integer, starting from 1

- **atom_1_name** – atom name
- **atom_2_res_index** – integer, starting from 1
- **atom_2_name** – atom name
- **r1** – in nanometers
- **r2** – in nanometers
- **r3** – in nanometers
- **r4** – in nanometers
- **k** – in $kJ/mol/nm^2$

class meld.system.restraints.**LinearScaler**(*alpha_min*, *alpha_max*, *strength_at_alpha_min=1.0*, *strength_at_alpha_max=0.0*)

This scaler linearly interpolates between 0 and 1 from alpha_min to alpha_max.

class meld.system.restraints.**NonLinearScaler**(*alpha_min*, *alpha_max*, *factor*, *strength_at_alpha_min=1.0*, *strength_at_alpha_max=0.0*)

class meld.system.restraints.**NonSelectableRestraint**

Abstract class for non-selectable restraints.

class meld.system.restraints.**RdcRestraint**(*system*, *scaler*, *atom_1_res_index*, *atom_1_name*, *atom_2_res_index*, *atom_2_name*, *kappa*, *d_obs*, *tolerance*, *force_const*, *weight*, *expt_index*)

Residual Dipolar Coupling Restraint

Parameters

- **system** – a System object
- **scaler** – a force scaler
- **atom_1_res_index** – integer, starting from 1
- **atom_1_name** – atom name
- **atom_2_res_index** – integer, starting from 1
- **atom_2_name** – atom name
- **kappa** – prefactor for RDC calculation in $Hz/Angstrom^3$
- **d_obs** – observed dipolar coupling in Hz
- **tolerance** – calculed couplings within tolerance (in Hz) of d_obs will have zero energy and force
- **force_const** – force sonstant in $kJ/mol/Hz^2$
- **weight** – dimensionless weight to place on this restraint
- **expt_index** – integer experiment id

Typical values for kappa are:

•1H - 1H: $-360300\ Hz/Angstrom^3$

•13C - 1H: $-90600\ Hz/Angstrom^3$

•15N - 1H: $36500\ Hz/Angstrom^3$

class meld.system.restraints.**Restraint**
Abstract class for all restraints.

class meld.system.restraints.**RestraintManager**(*system*)

class meld.system.restraints.**RestraintRegistry**(*name*, *bases*, *attrs*)
Metaclass that maintains a registry of restraint types.

All classes that decend from Restraint inherit RestraintRegistry as their metaclass. RestraintRegistry will automatically maintain a map between the class attribute '_restraint_key_' and all restraint types.

The function get_constructor_for_key is used to get the class for the corresponding key.

classmethod get_constructor_for_key(*key*)
Get the constructor for the restraint type matching key.

class meld.system.restraints.**Scaler**
Base class for all scalers.

class meld.system.restraints.**ScalerRegistry**(*name*, *bases*, *attrs*)
Metaclass that maintains a registry of scaler types.

All classes that decend from Scaler inherit ScalerRegistry as their metaclass. ScalerRegistry will automatically maintain a map between the class attribute '_scaler_key_' and all scaler types.

The function get_constructor_for_key is used to get the class for the corresponding key.

classmethod get_constructor_for_key(*key*)
Get the constructor for the scaler type matching key.

class meld.system.restraints.**SelectableRestraint**
Abstract class for selectable restraints.

class meld.system.restraints.**SelectivelyActiveCollection**(*restraint_list*, *num_active*)

class meld.system.restraints.**TorsionRestraint**(*system*, *scaler*, *atom_1_res_index*, *atom_1_name*, *atom_2_res_index*, *atom_2_name*, *atom_3_res_index*, *atom_3_name*, *atom_4_res_index*, *atom_4_name*, *phi*, *delta_phi*, *k*)
A torsion restraint

Parameters

- **system** – System
- **scaler** – force scaler
- **atom_1_res_index** – integer, starting from 1
- **atom_1_name** – atom name
- **atom_2_res_index** – integer, starting from 1
- **atom_2_name** – atom name
- **atom_3_res_index** – integer, starting from 1
- **atom_3_name** – atom name
- **atom_4_res_index** – integer, starting from 1
- **atom_4_name** – atom name
- **phi** – equilibrium value, degrees
- **delta_phi** – flat within delta_phi, degrees

- **k** – $kJ/mol/degree^2$

1.2.17 meld.system.runner

class `meld.system.runner.`**`FakeSystemRunner`**
Fake runner for test purposes.

1.2.18 meld.system.state

class `meld.system.state.`**`SystemState`**(*positions*, *velocities*, *alpha*, *energy*)
Class to hold the state of a system.

Parameters

- **positions** – coordinates of structure, `numpy.array(n_atoms, 3)`
- **velocities** – velocities for structure, same as coords
- **alpha** – alpha value, within `[0, 1]`
- **energy** – total potential energy, including restraints

1.2.19 meld.system.system

1.2.20 meld.util

`meld.util.`**`in_temp_dir`**(**args*, ***kwds*)
Context manager to run in temporary directory

1.2.21 meld.vault

class `meld.vault.`**`DataStore`**(*n_atoms*, *n_replicas*, *pdb_writer*, *block_size=100*)
Class to handle storing data from MELD runs.

Parameters

- **n_atoms** – number of atoms
- **n_replicas** – number of replicas
- **block_size** – size of netcdf blocks and frequency to do backups

Data will be stored in the 'Data' subdirectory. Backups will be stored in 'Data/Backup'.

Some information is stored as python pickled files:

•data_store.dat – the DataStore object

•communicator.dat – the MPICommunicator object

•remd_runner.dat – the MasterReplicaExchangeRunner object

Other data (positions, velocities, etc) is stored in the results.nc file.

`backup`(*stage*)
Backup all files to Data/Backup.

Parameters **stage** – int stage

Backup will occur if *stage % backup_freq == 0*

close()
Close the DataStore

initialize(*mode*)
Prepare to use the DataStore object.

Parameters **mode** – mode to open in.

Available modes are:

- 'w' – create a new directory structure and initialize the hd5 file
- 'a' – append to the existing files
- 'r' – open the file in read-only mode

iterate_permutation_vectors(*start=None*, *end=None*)
Iterate over the permutation vectors from disk.

iterate_positions(*start=None*, *end=None*)
Iterate over the positions from disk.

load_acceptance_probabilities(*stage*)
Load acceptance probability vector from disk.

Parameters **stage** – int stage to load

Returns n_replica_pairs array of int

load_all_acceptance_probabilities()
Load all acceptance probabilities from disk

Warning, this might take a lot of memory

load_all_alphas()
Load all alphas from disk.

Warning, this could use a lot of memory.

load_all_energies()
Load all energies from disk.

Warning, this could use a lot of memory

load_all_permutation_vectors()
Load all permutation vector from disk.

Warning, this might take a lot of memory

load_all_positions()
Load all positions from disk.

Warning, this could use a lot of memory.

load_all_velocities()
Load all velocities from disk.

Warning, this could use a lot of memory.

load_alphas(*stage*)
Load alphas from disk.

Parameters **stage** – int stage to load from disk

Returns n_replicas array

load_communicator()
Load the communicator from disk

classmethod load_data_store(*load_backup=False*)
Load the DataStore object from disk.

load_energies(*stage*)
Load energies from disk.

Parameters **stage** – int stage to load

Returns n_replicas array

load_permutation_vector(*stage*)
Load permutation vector from disk.

Parameters **stage** – int stage to load

Returns n_replicas array of int

load_positions(*stage*)
Load positions from disk.

Parameters **stage** – int stage to load

load_positions_random_access(*stage*)
Load positions from disk.

Parameters **stage** – int stage to load

This differs from `load_positions()` in that you can positions from any stage, while `load_positions()` can only move forward in time. However, this comes at a performance penalty.

load_remd_runner()
Load replica runner from disk

load_states(*stage*)
Load states from disk

Parameters **stage** – integer stage to load

Returns list of SystemState objects

load_velocities(*stage*)
Load velocities from disk.

Parameters **stage** – int stage to load

save_acceptance_probabilities(*accept_probs*, *stage*)
Save acceptance probabilities vector to disk.

Parameters

- **accept_probs** – n_replicas array of int
- **stage** – int stage to store

save_alphas(*alphas*, *stage*)
Save alphas to disk.

Parameters

- **alphas** – n_replicas array
- **stage** – int stage to store

save_communicator(*comm*)
Save the communicator to disk

save_data_store()
Save this object to disk.

save_energies(*energies*, *stage*)
Save energies to disk.

Parameters

- **energies** – n_replicas array
- **stage** – int stage to save

save_permutation_vector(*perm_vec*, *stage*)
Save permutation vector to disk.

Parameters

- **perm_vec** – n_replicas array of int
- **stage** – int stage to store

save_positions(*positions*, *stage*)
Save the positions to disk.

Parameters

- **positions** – n_replicas x n_atoms x 3 array
- **stage** – int stage to store

save_remd_runner(*runner*)
Save replica runner to disk

save_states(*states*, *stage*)
Save states to disk.

Parameters

- **states** – list of SystemStage objects to store
- **stage** – int stage to store

save_velocities(*velocities*, *stage*)
Save velocities to disk.

Parameters

- **velocities** – n_replicas x n_atoms x 3 array
- **stage** – int stage to store

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*

m