Supplementary Data for

gmxapi: a high-level interface for advanced control and extension of molecular dynamics simulations

M. Eric Irrgang, Jennifer M. Hays and Peter M. Kasson

Work specification grammar

Work is specified as structured data with minimal schema that are easily expressed with Python data types or serialized with JSON. For gmxapi 0.0.4, a valid work specification contains the key-value pair indicating the schema version and a node named “elements” containing a set of zero or more uniquely named work elements. An empty, but valid work specification, then, could be represented as the following in JSON.

```
{
    "version": "gmxapi_workspec_0_1",
    "elements": {}
}
```

An element of a work specification has four named properties. *`operation`* indicates the functionality represented by the element and maps to code accessible in the execution context. *`namespace`* indicates the provider of the operation, with standard operations provided by the API implementation in the `gmxapi` namespace. *`params`* holds an optional list of arguments to be provided to the operation, and dependencies on other named elements are listed in *`depends`*. To ensure that a work specification can be maintained in a valid state, elements may not be added unless all of the element’s dependencies are already fully specified.

Plugin Interface

Gmxapi insulates third party code from the internal `gmx::IMDModule` and `gmx::IForceProvider` library interfaces. Instead, forces calculated with external code implement a force and energy calculation in a framework for code with similar behavior and resource needs. To add a pair restraint that operates between two sites, the researcher creates a class that provides a `calculate()` public member function that takes two site locations as arguments. The calculation function for a pair restraint returns a gmx::PotentialPointData object, which is a simple container for a force and energy. During the MD simulation, the Restraint framework feeds coordinates for the user-selected sites to the calculation function and applies the returned force appropriately.

Wrappers and a Python interface are generated when C++ templates are instantiated in a pybind11 export call for

`PyRestraint<RestraintModule<MyCustomRestraint>>`. Additional boiler plate code can be copied from a sample restraint, but is being reduced as the templates are migrated to the core package.

According to the registration/binding procedure, a plugin object exists before the simulation begins and after it ends. By providing a non-trivial builder or launcher according to the work execution protocol, the module can retain access to resources originally provided through the Python interpreter, such as other extension code or references to Python resources.

Restrained-ensemble simulation formalism

We have implemented a variant of the restrained-ensemble simulation formalism described previously by Roux and co-workers (Islam, et al., 2013; Roux and Islam, 2013; Roux and Weare, 2013). Our implementation is designed to account for substantial variation in protein backbone conformation between ensemble members, so we run MD simulations of 20 independent ensemble members. Distance histograms are estimated from this ensemble and updated at regular intervals (here every 100 ps and computing histograms over a 10-ns window). The biasing force for each residue-residue pair where an experimental bias is applied is calculated from the estimated distance histogram and the experimental distance histogram as originally derived by Roux (Roux and Islam, 2013) and described below.

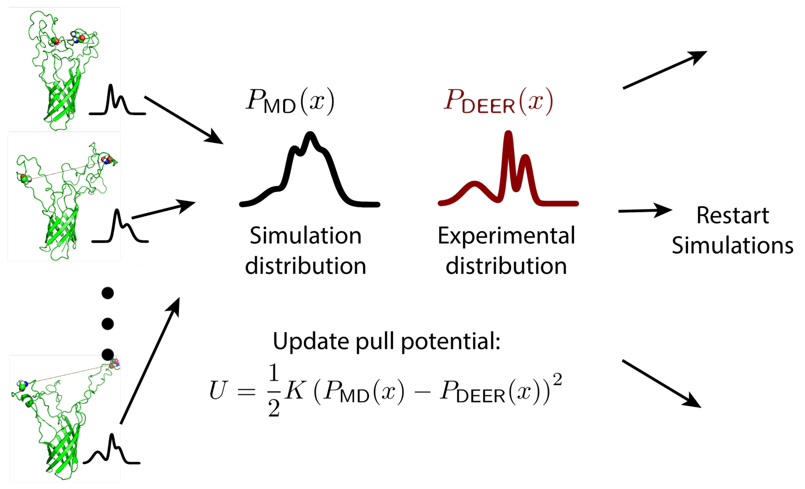

Figure S1. Schema for restrained-ensemble simulations.

At every potential update step, the simulation distribution for pair *ij* is smoothed with a differentiable Gaussian of width σ as follows:

$$h^{ij}(n) = \frac{1}{N}\sum_{\tau=1}^{M}\sum_{s=1}^{N}\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\left(n\Delta r-\left|\boldsymbol{r}_i^{s,\tau}-\boldsymbol{r}_j^{s,\tau}\right|\right)^2/2\sigma^2}$$

where *n* is the bin number, *N* is the number of ensemble members, *M* is the total number of samples within the boxcar averaging window of choice, and $\left|r_i^{s,\tau} - r_j^{s,\tau}\right|$ denotes the distance between residues *i* and *j* in simulation *s* at time τ. A new potential is then applied:

$$U_{RE} = \frac{1}{2} K \sum_{\{\text{pair } ij\}} \sum_{\{n\}} \left(h^{ij}(n) - H^{ij}(n)\right)^2$$

where $H^{ij}(n)$ is the smoothed experimentally-derived distance distribution for pair *ij*. This potential is applied for an interval Δt until another update step takes place.

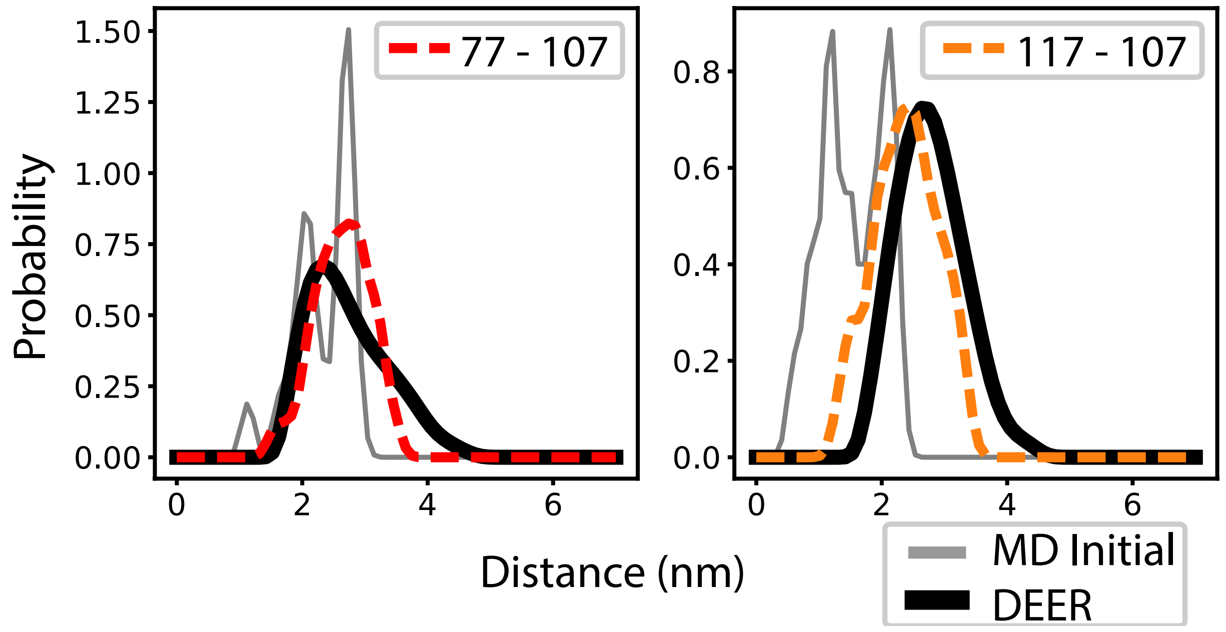

Figure S2. Restrained-ensemble fitting to DEER data. Two residue-residue distance distributions for the protein Opa60 determined experimentally via DEER are plotted in black, together with initial distributions for a conformational ensemble in gray and the results of restrained-ensemble fitting with 20 ensemble members each simulated for 100ns in dotted lines. The restrained-ensemble fitting improves agreement with both sets of experimental DEER data simultaneously. The starting structures are taken from the previous NMR/MD hybrid refinement of this protein, Opa$_{60}$ (Fox, et al., 2014). Simulations were performed using the CHARMM36 force field (Huang and MacKerell, 2013), run under NPT conditions at 310K with the velocity-rescaling themostat (Bussi, et al., 2007)and Parrinello-Rahman barostat (Parrinello and Rahman, 1982) with 2-ps and 10-ps time constants respectively. Long-range electrostatics were treated with Particle Mesh Ewald (Darden, et al., 1993).

Gmxapi implementation of restrained-ensemble formalism

The gmxapi sample restraint plugin provides a restraint that can be used to perform the above work. The relevant lines of Python follow below. Note that the sample plugin is assumed to be in the Python module path and named “myplugin” and for brevity we assume the existence of an array of input filenames and of restraint parameters.

Source for the plugin code can be found at:
https://github.com/kassonlab/sample_restraint

```
import gmx
import myplugin

# Specification of run input files for the ensemble members
n_ensemble = 20
input_dir_list = ['run_{:02d}'.format(i) for i in range(n_ensemble)]
tpr_list = [os.path.join(directory, 'run.tpr') for directory in input_dir_list]

# Parameters used in the restrained-ensemble simulations
index1 = 387  # index of first atom to restrain
index2 = 2569  # index of second atom to restrain
window_size = 5000 * 0.002  # time interval between potential updates in ps
output_interval = 1000 * 0.002  # time interval to output forces and positions
k = 100.  # force constant
sigma = 0.2
nbins = 70  # number of bins in histogram
binwidth = 0.1  # bin width in nm
maxdistance = 6.0  # maximum distance in nm
mindistance = 1.9  # minimum distance in nm
histogram = [0.05] * nbins

# Set up and run
params = [index1, index2, nbins, mindistance, maxdistance,
          histogram, window_size / output_interval, output_interval, n_ensemble,
          window_size, k, sigma]

potential = gmx.workflow.WorkElement(
        namespace="myplugin",
        operation="ensemble_restraint",
        depends=[],
        params=params
        )
potential.name = "ensemble_restraint_1"

md = gmx.workflow.from_tpr(tpr_list)
md.add_dependency(potential)

context = gmx.context.ParallelArrayContext(md)

with context as session:
        session.run()
```

A work specification for the restrained-ensemble workflow then looks like the following JSON object, where the load_tpr params holds a list of absolute file paths.

```
    {
        "version": "gmxapi_workspec_0_1",
        "elements":
        {
            “tpr_input”:
            {
                “namespace”: “gmxapi”,
                “operation”: “load_tpr”,
                “params”: […],
                “depends”: []
            }
            “md_sim”:
            {
                “namespace”: “gmxapi”,
                “operation”: “md”,
                “params”: [],
                “depends”: [“tpr_input”, “ensemble_restraint”]
            }
            “ensemble_restraint_1”:
            {
                “namespace”: “myplugin”,
                “operation”: “ensemble_restraint”,
                “params”: […],
                “depends”: []
            }
        }
    }
```

Instead of reading the experimental distribution directly from a file, the array of values and the histogram parameters are provided to the ensemble restraint work element parameters. The working histograms are generated in memory and updated with the help of an ensemble reduce operation, which appears as an additional downstream node in the execution graph, generated by the ensemble_restraint builder. In future versions of gmxapi, such a Context-provided resource may be expressed in the higher-level work specification.

References:

Bussi, G., Donadio, D. and Parrinello, M. Canonical sampling through velocity rescaling. *J Chem Phys* 2007;126(1):014101.

Darden, T., York, D. and Pedersen, L. Particle Mesh Ewald - an N.Log(N) Method for Ewald Sums in Large Systems. *Journal of Chemical Physics* 1993;98(12):10089-10092.

Fox, D.A.*, et al.* Structure of the Neisserial outer membrane protein Opa(6)(0): loop flexibility essential to receptor recognition and bacterial engulfment. *J Am Chem Soc* 2014;136(28):9938-9946.

Huang, J. and MacKerell, A.D., Jr. CHARMM36 all-atom additive protein force field: validation based on comparison to NMR data. *J Comput Chem* 2013;34(25):2135-2145.

Islam, S.M.*, et al.* Structural refinement from restrained-ensemble simulations based on EPR/DEER data: application to T4 lysozyme. *J Phys Chem B* 2013;117(17):4740-4754.

Parrinello, M. and Rahman, A. Strain Fluctuations and Elastic-Constants. *Journal of Chemical Physics* 1982;76(5):2662-2666.
Roux, B. and Islam, S.M. Restrained-ensemble molecular dynamics simulations based on distance histograms from double electron-electron resonance spectroscopy. *J Phys Chem B* 2013;117(17):4733-4739.
Roux, B. and Weare, J. On the statistical equivalence of restrained-ensemble simulations with the maximum entropy method. *J Chem Phys* 2013;138(8):084107.