Towards MDP.jl: The Julia Library of MD Potentials

April 5, 2021

Overview

My participation in CESMIX...

- The sub-projects in which I am involved regarding the CESMIX project are:
  - Implementing a Julia version of the "descriptors" and "force calculation" of MDP.jl
  - Collaborating with the integration between NBodySimulator.jl with DFTK.jl and MDP.jl
- I am interacting with Alan Edelman, Ngoc Cuong Nguyen, Jeremiah R DeGreeff, Michael F. Herbst, Valentin Churavy, Andrew Rohskopf, Dionysios Sema, and Mathew M. Swisher.
- I am preparing a presentation about my progress:
  - I will collaborate with Cuong in the upcoming CESMIX presentation (May 18).
  - I will collaborate with Alan and Jeremiah in the upcoming CESMIX presentation (June 15).
  - I will give a talk in JuliaCon2021 about the latest developments in MDP.jl (July 28)

Note: this slide will not be part of the the JuliaCon2021 presentation.

MDP.jl: the Julia library of MD potentials

- **Molecular Dynamics (MD) simulations require a potential energy function** to describe the force field governing the interaction among atoms.

MDP.jl: the Julia library of MD potentials

- **Molecular Dynamics (MD) simulations require a potential energy function** to describe the force field governing the interaction among atoms.
- Force calculation often takes between 50% and 90% of the overall complexity, thus, **determining an adequate potential is a crucial task**.

MDP.jl: the Julia library of MD potentials

- **Molecular Dynamics (MD) simulations require a potential energy function** to describe the force field governing the interaction among atoms.
- Force calculation often takes between 50% and 90% of the overall complexity, thus, **determining an adequate potential is a crucial task**.
- MDP.jl will provide **fast and accurate potentials for classical MD simulations on exascale supercomputers.**
  - Coupling empirical and machine learning (ML) potentials
  - Quantifying uncertainties in trained ML potentials.
  - Identifying near-optimal configurations to include in the training data.

Potential energy fuctions

- Potential energy fuctions are determined based on **quantum information**, e.g. Density Functional Theory (DFT) data.

Potential energy fuctions

- Potential energy fuctions are determined based on **quantum information**, e.g. Density Functional Theory (DFT) data.
- Classical MD simulations use
  - **Empirical Potentials** (EP), such as Lennard-Jones or Tersoff.
  - **Machine Learning Potentials** (MLP), such as Neural Network Potentials (NNP) or Spectral Neighborhood Analysis Potentials (SNAP).

Potential energy fuctions

- Potential energy fuctions are determined based on **quantum information**, e.g. Density Functional Theory (DFT) data.
- Classical MD simulations use
  - **Empirical Potentials** (EP), such as Lennard-Jones or Tersoff.
  - **Machine Learning Potentials** (MLP), such as Neural Network Potentials (NNP) or Spectral Neighborhood Analysis Potentials (SNAP).
- **Present limitations** when solving complex study cases as ultrahigh temperature ceramics in hypersonic flows.
  - Existing EP, ReaxFF and COMB, do not produce satisfactory results since they **require retraining.**
  - MLP **demand a significant amount of** DFT **data** for the training process.

Coupling Empirical and ML potentials

- Coupling both potentials is a promising approach for
  - **Reducing the amount of training data**
  - **Leveraging physics information coded in the EP**

Coupling Empirical and ML potentials

- Coupling both potentials is a promising approach for
  - **Reducing the amount of training data**
  - **Leveraging physics information coded in the EP**
- Optimization problem

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}\in\mathbb{R}^M} \sum_{j=1}^{J} w_j \sum_{i=1}^{N_j} \left| \boldsymbol{f}(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i) - \boldsymbol{f}^{\mathrm{qm}}(\boldsymbol{r}^{N_j}, j, i) \right|^2 .$$

Coupling Empirical and ML potentials

- Coupling both potentials is a promising approach for
  - **Reducing the amount of training data**
  - **Leveraging physics information coded in the EP**
- Optimization problem

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}\in\mathbb{R}^M} \sum_{j=1}^{J} w_j \sum_{i=1}^{N_j} \left| \boldsymbol{f}(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i) - \boldsymbol{f}^{\mathrm{qm}}(\boldsymbol{r}^{N_j}, j, i) \right|^2 .$$

$$\boldsymbol{f}(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i) = \boldsymbol{f}_E(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i) + \boldsymbol{f}_{ML}(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i)$$

Empirical component of the force

Empirical component of the force:

$$\boldsymbol{f}_E(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i) = \boldsymbol{f}_{E,PS}(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i) + \boldsymbol{f}_{E,BS}(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i)$$

Empirical component of the force

Empirical component of the force:

$$\boldsymbol{f}_E(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i) = \boldsymbol{f}_{E,PS}(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i) + \boldsymbol{f}_{E,BS}(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i)$$

The **power spectrum** component (long range interaction):

$$\boldsymbol{f}_{E,PS}(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i) = \sum_{t=1}^{N_z} \sum_{k=1}^{K} \sum_{k'=k}^{K} \sum_{l=0}^{L} c_{tkk'l} \frac{\partial d_{tkk'l}^{PS}(\boldsymbol{r}^{N_j}, j, i)}{\partial \boldsymbol{r}_i^{N_j}}$$

Empirical component of the force

Empirical component of the force:

$$\boldsymbol{f}_E(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i) = \boldsymbol{f}_{E,PS}(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i) + \boldsymbol{f}_{E,BS}(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i)$$

The **power spectrum** component (long range interaction):

$$\boldsymbol{f}_{E,PS}(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i) = \sum_{t=1}^{N_z} \sum_{k=1}^{K} \sum_{k'=k}^{K} \sum_{l=0}^{L} c_{tkk'l} \frac{\partial d_{tkk'l}^{PS}(\boldsymbol{r}^{N_j}, j, i)}{\partial \boldsymbol{r}_i^{N_j}}$$

The **bispectrum** component (short range interactions):

$$\boldsymbol{f}_{E,BS}(\boldsymbol{r}^{N_j}, \boldsymbol{c}, j, i) = \sum_{t=1}^{N_z} \sum_{k=1}^{K} \sum_{k'=k}^{K} \sum_{l=0}^{L} \sum_{l_1=0}^{L} \sum_{l_2=0}^{L} c_{tkk'll_1l_2} \frac{\partial d_{tkk'll_1l_2}^{BS}(\boldsymbol{r}^{N_j}, j, i)}{\partial \boldsymbol{r}_i^{N_j}}$$

Empirical potential: PS basis functions

Derivative of the power spectrum basis functions

$$\frac{\partial d^{BS}_{tkk'l}(\boldsymbol{r}^{N_j}, j, i)}{\partial \boldsymbol{r}_i^{N_j}} = \sum_{s \in \Omega'_{jit}} p^{\partial}_{iskk'l}(\boldsymbol{r}^{N_j}, j) - \sum_{s \in \Omega''_{jit}} p^{\partial}_{sikk'l}(\boldsymbol{r}^{N_j}, j),$$

$$p^{\partial}_{i_0 i_1 kk'l}(\boldsymbol{r}^{N_j}, j) = \sum_{m=-l}^{l} \left(\frac{\partial u_{klm}(\boldsymbol{r}_{i_0}^{N_j} - \boldsymbol{r}_{i_1}^{N_j})}{\partial(\boldsymbol{r}_{i_0}^{N_j} - \boldsymbol{r}_{i_1}^{N_j})} \sum_{s \in \Omega_{j,i_1}} \left(u_{k'lm}(\boldsymbol{r}_s^{N_j} - \boldsymbol{r}_{i_1}^{N_j}) \right) \right) +$$

$$\sum_{m=-l}^{l} \left(\frac{\partial u_{k'lm}(\boldsymbol{r}_{i_0}^{N_j} - \boldsymbol{r}_{i_1}^{N_j})}{\partial(\boldsymbol{r}_{i_0}^{N_j} - \boldsymbol{r}_{i_1}^{N_j})} \sum_{s \in \Omega_{j,i_1}} \left(u_{klm}(\boldsymbol{r}_s^{N_j} - \boldsymbol{r}_{i_1}^{N_j}) \right) \right)$$

Empirical potential: PS basis functions

- The descriptors are based on products of radial basis functions $g_{lk}(r)$ and spherical harmonics $Y_{lm}(\theta, \phi)$.

$$u_{klm}(\boldsymbol{r}) = g_{lk}(r) Y_{lm}(\theta, \phi)$$

Empirical potential: PS basis functions

- The descriptors are based on products of radial basis functions $g_{lk}(r)$ and spherical harmonics $Y_{lm}(\theta, \phi)$.

$$u_{klm}(\boldsymbol{r}) = g_{lk}(r) Y_{lm}(\theta, \phi)$$

- $g_{lk}(r)$ can vary...
  - Spherical Bessel
  - Polynomials, Gaussian functions, etc.

Empirical potential: PS basis functions

- The descriptors are based on products of radial basis functions $g_{lk}(r)$ and spherical harmonics $Y_{lm}(\theta, \phi)$.

$$u_{klm}(\boldsymbol{r}) = g_{lk}(r) Y_{lm}(\theta, \phi)$$

- $g_{lk}(r)$ can vary...
  - Spherical Bessel
  - Polynomials, Gaussian functions, etc.
- Providing flexibility when calculating the derivative of $u_{klm}(\boldsymbol{r})$ using the finite difference method (FDM)

$$\frac{\partial u_{klm}(\boldsymbol{r})}{\partial(\boldsymbol{r})} = \left(\frac{u_{klm}(\boldsymbol{r} + \boldsymbol{\Delta} x) - u_{klm}(\boldsymbol{r} - \boldsymbol{\Delta} x)}{2|\boldsymbol{\Delta} x|}, ..., ... \right)$$

Empirical potential: BS basis functions

Derivative of the bispectrum basis functions

$$\frac{\partial d^{BS}_{tkk'll_1l_2}(\boldsymbol{r}^{N_j}, j, i)}{\partial \boldsymbol{r}_i^{N_j}} = \left(\frac{d_{tkk'll_1l_2}(\boldsymbol{r}^{N_j} + \boldsymbol{\Delta} X_i, j, i) - d_{tkk'll_1l_2}(\boldsymbol{r}^{N_j} - \boldsymbol{\Delta} X_i, j, i)}{2|\boldsymbol{\Delta} X_i|}, \ldots, \ldots \right)$$

Empirical potential: BS basis functions

Derivative of the bispectrum basis functions

$$\frac{\partial d^{BS}_{tkk'll_1l_2}(\boldsymbol{r}^{N_j}, j, i)}{\partial \boldsymbol{r}_i^{N_j}} = \left(\frac{d_{tkk'll_1l_2}(\boldsymbol{r}^{N_j} + \boldsymbol{\Delta} X_i, j, i) - d_{tkk'll_1l_2}(\boldsymbol{r}^{N_j} - \boldsymbol{\Delta} X_i, j, i)}{2|\boldsymbol{\Delta} X_i|}, \ldots, \ldots \right)$$

The bispectrum basis functions are formulated as:

$$d^{BS}_{tkk'll_1l_2}(\boldsymbol{r}^{N_j}, j, i) = \sum_{s \in \Omega'''_{jt}} b_{skk'll_1l_2}(\boldsymbol{r}^{N_j}, j, i)$$

$$b_{skk'll_1l_2}(\boldsymbol{r}^{N_j}, j, i) = \sum_{m=-l}^{l} \sum_{m_1=-l_1}^{l_1} \sum_{m_2=-l_2}^{l_2} \bar{a}_{sklm}(\boldsymbol{r}^{N_j}, j) C^{l_1l_2l}_{m_1m_2m} a_{sk'l_1m_1}(\boldsymbol{r}^{N_j}, j) a_{sk'l_2m_2}(\boldsymbol{r}^{N_j}, j)$$

$$a_{iklm}(\boldsymbol{r}^{N_j}, j) = \sum_{s \in \Omega_{ji}} u_{klm}(\boldsymbol{r}_s^{N_j} - \boldsymbol{r}_i^{N_j})$$

Next steps

- Keep my summary of implemented equations up to date ✓

Next steps

- Keep my summary of implemented equations up to date ✓
- Optimization
  - Clebsch–Gordan coefficients
    - Fast calculation using "PartialWaveFunctions.jl" ✓
    - Stop calculation when a zero is found ✓
  - Calculate only in the neighbors
    - Precalculated neighbors information ✓
    - Optimize precalculation
  - Leverage spherical harmonics symetries
  - Threading and GPU

Next steps

- Keep my summary of implemented equations up to date ✓
- Optimization
  - Clebsch–Gordan coefficients
    - Fast calculation using "PartialWaveFunctions.jl" ✓
    - Stop calculation when a zero is found ✓
  - Calculate only in the neighbors
    - Precalculated neighbors information ✓
    - Optimize precalculation
  - Leverage spherical harmonics symetries
  - Threading and GPU
- Testing: the descriptors are invariant to
  - Rotation ✓
  - Permutation
  - Translation

Next steps

- Keep my summary of implemented equations up to date ✓
- Optimization
  - Clebsch–Gordan coefficients
    - Fast calculation using "PartialWaveFunctions.jl" ✓
    - Stop calculation when a zero is found ✓
  - Calculate only in the neighbors
    - Precalculated neighbors information ✓
    - Optimize precalculation
  - Leverage spherical harmonics symetries
  - Threading and GPU
- Testing: the descriptors are invariant to
  - Rotation ✓
  - Permutation
  - Translation
- Use automatic differentiation packages in Julia.

Next steps

- Keep my summary of implemented equations up to date ✓
- Optimization
  - Clebsch–Gordan coefficients
    - Fast calculation using "PartialWaveFunctions.jl" ✓
    - Stop calculation when a zero is found ✓
  - Calculate only in the neighbors
    - Precalculated neighbors information ✓
    - Optimize precalculation
  - Leverage spherical harmonics symetries
  - Threading and GPU
- Testing: the descriptors are invariant to
  - Rotation ✓
  - Permutation
  - Translation
- Use automatic differentiation packages in Julia.
- ML potentials

Next steps

- Keep my summary of implemented equations up to date ✓
- Optimization
  - Clebsch–Gordan coefficients
    - Fast calculation using "PartialWaveFunctions.jl" ✓
    - Stop calculation when a zero is found ✓
  - Calculate only in the neighbors
    - Precalculated neighbors information ✓
    - Optimize precalculation
  - Leverage spherical harmonics symetries
  - Threading and GPU
- Testing: the descriptors are invariant to
  - Rotation ✓
  - Permutation
  - Translation
- Use automatic differentiation packages in Julia.
- ML potentials
- Integration with NBodysimulator and LAMMPS

Next steps

- Keep my summary of implemented equations up to date ✓
- Optimization
  - Clebsch–Gordan coefficients
    - Fast calculation using "PartialWaveFunctions.jl" ✓
    - Stop calculation when a zero is found ✓
  - Calculate only in the neighbors
    - Precalculated neighbors information ✓
    - Optimize precalculation
  - Leverage spherical harmonics symetries
  - Threading and GPU
- Testing: the descriptors are invariant to
  - Rotation ✓
  - Permutation
  - Translation
- Use automatic differentiation packages in Julia.
- ML potentials
- Integration with NBodysimulator and LAMMPS
- Compare C++ implementation about descriptors and force calculation with this Julia implementation.

MDP.jl + NBodySimulator.jl

Argon study case.

1. Run MDP.jl:
   - Input (from the initial condition):
     - Force field (calculated through LJ, thus, the initial force field). Atomic positions.
   - Output:
     - Force field (this fitted function will be used in the whole MD simulation)
2. Run NBodySimulator.jl

   For each time step:
   - Input (from the initial condition):
     - Force field from MDP.jl (the fitted force field expression is evaluated here). Atomic Velocities. Domain(box). Temperature. Mass. External forces.
   - Output:
     - Atom positions. Atomic velocities. Temperature. Energy (kinetic, potential, and total). Radial distribution function

DFTK.jl + NBodySimulator.jl

Argon study case.

1. Run NBodySimulator.jl

   For each time step:

   - Input: Quantum force from DFTK.jl
     - DFTK.jl input: Atomic positions from NBodySimulator.jl
     - DFTK.jl output: Quantum force (a single set of forces evaluated at one set of positions). Atomic positions. Atomic velocities. Domain (box). Temperature. Mass. External forces
   - Output: Atom positions. Atomic velocities. Temperature. Energy (kinetic, potential, and total). Radial distribution function.

References

- "MDP.jl: The Julia Library of Molecular Dynamics Potentials". MIT PSAAP-3 Team. 2021.
- "Accelerating Force Calculation in Molecular Dynamics with Transition to LAMMPS". MIT PSAAP-3 Team. 2021.
- MDP. https://github.com/cesmix-mit/MDP.jl
- Fortran and Ar MD simulation. https://ase.tufts.edu/chemistry/lin/outreach_FortranMD.html
- NBodySimulator. https://github.com/SciML/NBodySimulator.jl
- Molly. https://github.com/JuliaMolSim/Molly.jl
- Argon study case using NBodySimulator. https://github.com/jrdegreeff/cesmix-julia/blob/main/notebooks/nbodysimulator_argon_simulation.jl
- Draft DFTK+Molly / MDP+Molly. https://docs.google.com/document/d/1APQ1JjL0SoQTTsDam5j3lEKe3xCDwNzrbRIGNH_JzxU/edit?usp=sharing

Thanks :-)