Research Outline

Andrew White

July 9, 2013

Research Strengths: programming, probability, machine learning. Peptides, experiments, materials. Graphics (2D/3D), cloud computing.

Goal 1: Develop new algorithms for rigorous and practical coupling between experiments and simulations.

Goal 2: Work on non-algorithm non-bio project

Goal 3: Become familiar with coarse-graining techniques in Voth group

1 Connecting Experiments and Simulations

I see two approaches for accomplishing this and both dovetail nicely with other algorithm challenges in molecular simulations.

1.1 Extending Force-matching

The first approach is to use an optimization scheme like force matching to improve an existing force field given experimental data. In force matching, one optimizes $\sum_i (F_{target} - F_{ref})^2$. However, one could match something with regularization, constraints or other loss terms. For example, $w_1 \sum_i (F_{target} - F_{ref})^2 + w_2(< \rho >_{target} - \rho_{measured})^2$. This would allow one to coarse-grain while matching density. Alternatively, the topology can be left untouched and the force field updated to more closely match the density while perturbing the frame-by-frame force deviation as little as possible. Checking the constraint of density would use reweighting trajectory frames.

The challenge here, aside from traditional reweighting challenges, is to find a robust alternative to linear-least squares. The non-spline terms will not have constant derivatives. Stochastic gradient descent is one approach that is simple, robust, and routinely applied to problems just like the one described. There are other options as well though. Developing this capability would also allow new classes of coarse-grain potential functions and possibly improve transferability since the number of parameters and we could enforce regularization to prevent over-fitting during coarse-graining.

I'm currently writing code to test this with a Lennard-Jones fluid simulation engine. No results to report yet. I've also been looking through the literature on anything related to this.

1.2 Online Gradient Descent

Assume we have a basis set of kernel functions, $K_k(r_{ij})$, where k is in the index of the basis function and r_{ij} is the distance between the ith and jth particles. The force experienced between particles i and j is $\sum_k^M w_k K_k(r_{ij}) n_{ij}$. M is the number of basis functions, w_k is the weight of the kth basis function, and n_{ij} is the normal vector beween the two particles. The loss function is:

$$l = \frac{1}{3N} \left| \sum_i^N \left[F_i - \sum_j^{N_i} \sum_k^M w_k K_k(r_{ij}) n_{ij} \right] \right|^2$$

where F_i is the reference force on particle i, N_i is the number of neighbors of particle i and N is the number of particles. The gradient of the rhs inner term is

$$\sum_i^N \sum_j^{N_i} K_{k'}(r_{ij}) n_{ij}$$

The gradient of the loss function may be calculated using that quantity. The update rule is:

$$w' = w - \eta \partial l$$

The kernal functions may be anything. For now, I'll just use step functions.

Code outline. Need: algorithm, target forces, geoemtry, kernel functions. Target forces – symbpy. Class for pairwise and bonded. Start with pairwise. Geometry – xyz reader. Kernel functions: do not use symbpy. Too slow. Algorithm – see if we can do it with numpy. Pay attention to the convexity of your weights.

Plan for implementation: a) Create force matching class that takes trajectory file. It will initialize weights, load file. Has collection of reference force category classes (pairwise). Each category contains one or more force evaluations and returns a force given a trajectory and particle index. Force mathcing class also takes basis function categories. These return the basis value vector given the trajcetory and index. That vector can then be multiplied with the weight vector. Let's go for constant learning rate for now. Also give the class some useful plotting routines

b) Target force things are implemented in smypy.

c) Basis functinos are created with smypy

d) That's it!

I need to rework my plan a little bit. I would like to completely separate the algorithm from the MD code. My toolchian is going to MDAnalaysis, that KDTree code I have starred on git, numpy, smypy, and matplotlib.

OMG OMG OMG OMG! I figured it out! Here's how we do property matching!

The loss function is $< f(x) >$ where $f(x)$ is some error in a simulation relative to an observable, for example $< f(x) - \hat{f} >$. Or squared error, or whatever. We want to minimize that value by changing our energy function to weight low values higher and high values lower. For example, if we observed $f(x_0) = 3$, $f(x_1) = 2$, $f(x_2) = 0.01$, then we'd trying to make the energy lowest in the x_2 configuration. To accomplish this minimization, we need a gradient wrt to the energy of our loss funciton. This may be calculated as follows:

$$\frac{\partial < f(x) >}{\partial w_k} = \frac{\partial}{\partial w_k}\frac{1}{Z}\sum_i f(x_i)e^{-\beta U(x_i)}$$

$$= \frac{1}{Z}\sum_i -\beta\frac{\partial U(x_i)}{\partial w_k}f(x_i)e^{-\beta U(x_i)} - \frac{1}{Z^2}\frac{\partial Z}{\partial w_k}\sum_i f(x_i)e^{-\beta U(x_i)}$$

$$= -\left\langle \beta\frac{\partial U(x)}{\partial w_k}f(x)\right\rangle - \frac{1}{Z}\sum_i -\beta\frac{\partial U(x_i)}{\partial w_k}e^{-\beta U(x_i)}\left\langle f(x_i)\right\rangle$$

$$= \left\langle \beta\frac{\partial U(x_i)}{\partial w_k}\right\rangle \left\langle f(x_i)\right\rangle - \left\langle \beta\frac{\partial U(x_i)}{\partial w_k}f(x)\right\rangle$$

$$= -Cov(\beta\frac{\partial U(x_i)}{\partial w_k}, f(x_i))$$

So, I can approximate this derviatve by sampling pairs of f(x) and the U derivative. However, the expecation/covariance is over the current potential, not the potential that the simulation was conducted. Luckily, I can use importance sampling. I do the same approach, sampling these pairs, and then I reweight them with the ratio of probabilities. Of course, I don't have the normalizing constant though. Huh. Well. shit. I guess I could just do bursts. Like a few frames at a time. That sounds fine. The other alteranative is to do an MCMC over frames. That's silly. So then, the importance sampling algorithm is:

$$Cov(\beta\frac{\partial U(x_i)}{\partial w_k}, f(x_i))...$$

You know it, I don't need to write it down.

1.3 Biased approach

The second approach, more along the lines of my proposed research in the fellowships applications is to work with biases on an MD simulation. The math here is more difficult, part of the reason that only toy systems have been explored, but I think it's possible and just as important. The interesting connection here is that if the biases can be time-dependent, one could use the approach to do reverse coarse-graining where the coarse-grained system is treated as an external bias on the all-atom MD.

I've worked some before arriving in Chicago on equations for this. If you think it's worth the time, I'll implement and test them.

2 Non-bio project

Let's discuss more, but I would like to work on an applied project that is in material science or energy if feasible.

3 Other

James and I talked about updating the force-matching code to be more general so that both new optimization algorithms (as mentioned above) and potential equations can be easily tested. I'd do this by porting most of the code to python where there exists an algebra system that can work with equations directly thereby eliminating the time-consuming process of coding every force field function term and derivatives.