Molecular Dynamics, Digital Material, and Design Patterns
Physics 7682 / CIS 6229: Computational Methods for Nonlinear Systems

- **Molecular dynamics**
  - Integration of Newton’s 2nd law for a large collection of particles
    \[ \vec{F} = m \vec{a} \]

- **physics**
  - thermodynamics of (non)interacting gases
  - structure, energetics, and dynamics of liquids, crystalline solids and defects (e.g., dislocations, cracks), disordered systems (e.g., glasses)
  - interatomic potentials, increasingly derived from quantum mechanical calculations

- **biology**
  - structure, energetics, and dynamics of macromolecules (e.g., proteins)
  - details of protein conformations, protein-ligand binding
  - interatomic potentials typically empirically derived (AMBER, CHARMM, etc.)
Molecular Dynamics module

- Not usual hints+fill-in-the-missing code structure
  - focused more on use of existing package (Digital Material) and analysis of simulation data
  - uses VPython (visual) for animated graphics

- Exercises
  - Perfume Walk: random walk due to collisions
  - Pressure: emergence of pressure from collisions
  - Equilibration: convergence to equilibrium from nonequilibrium state
  - Exponential Atmosphere: thinning of atmosphere under gravity
  - Pair Distribution Function: positional correlations in gas, liquid, solid

![Perfume Walk](image1.png)
![Pressure](image2.png)
![Equilibration](image3.png)
![Pair Distribution](image4.png)
![Exponential Atmosphere](image5.png)
Digital Material package

- Grew out of a multiscale materials modeling effort (Sethna, Myers, et al. 1998-2002)
  - emphasis on investigating the structure and dynamics of defects in crystalline and polycrystalline materials relevant for deformation and failure
    - dislocations, dislocation tangles, grain boundaries, cracks
  - emphasis on construction and manipulation on nontrivial, heterogeneous defect geometries and coupling to coarse-grained models (e.g., finite-element analysis)
  - original code in C++ with minimal Python wrapper; ported to Python for Phys 682 / CIS 629 by Sethna
Class Diagram: interactions among classes in OO code

Diagram shows the relationships between classes and their methods. The diagram includes classes such as EnergyObserver, BoundaryCondition, NeighborLocator, ListOfAtoms, Constraint, Transformer, and others. Each class has its own set of methods and relationships with other classes through associations, interfaces, and multiplicities.
Class Diagram: interactions among classes in OO code

Design patterns

• Object-oriented design methodology for building code that can easily incorporate changes
  - collaborations among sets of classes/objects to encapsulate highly variable pieces
  - different patterns encapsulate different types of variability, e.g.,
    - factories address variability in the construction of objects
    - observers address variability in views/analyses of a central data repository
    - strategies address variability in algorithms
    - adapters address variability in class/object interfaces
  - emphasis on having many smaller objects working together (more reconfigurable), rather than more specialized and monolithic pieces that are harder to change
  - initially catalogued in the well-known book by Gamma et al. (“Gang of Four”, of GoF)

• Design patterns in Digital Material
  - ListOfAtoms: efficient collections of atoms
  - Initializers and transformers: decoupled algorithms for manipulating structure
  - Movers: decoupling structure from dynamics (use different algorithms)
  - Observers: decoupling analysis from dynamics (generic Update() interface)
  - Boundary conditions: decoupling enforcement of boundary conditions from dynamics
  - NeighborLocators: decoupling structure from force computation
Gravity cluster

```python
gravityPotential = GravityPotential(g=g)
LennardJonesPotential = LennardJonesCutPotential()
potential = CompositePotential([gravityPotential,
                                 LennardJonesPotential])
boundaryConditions = ReflectiveBoundaryConditions(L)
neighborLocator = SimpleNeighborLocator(LennardJonesPotential.cutoff,
                                         boundaryConditions)
atoms = TriangularSphericalClusterListOfAtoms(
    R=R, center=[L/2., L/2.], temperature=T,
    radius=LennardJonesPotential.latticeSpacing/2.0)
displayObserver = VisualDisplayAtomsObserver(atoms,L)
energyObserver = EnergyObserver(potential, neighborLocator,
                                boundaryConditions)
observers = [displayObserver, energyObserver]
mover = RunVelocityVerlet;
sys = MDSystem(L, atoms, observers, neighborLocator,
                boundaryConditions, potential, mover)
```