

Polymer chain Dynamics. Simulation Framework- user manual

Ofir Shukron

April 6, 2015

Contents

0.1	Simulation Framework	1
1	Classes	2
1.1	RouseSimulatorFramework	2
1.1.1	Properties	2
1.1.2	Methods	3
1.2	SimulationDataRecorder	3
1.2.1	Properties	3
1.2.2	Methods	4
1.3	Rouse	4
1.3.1	Properties	4
1.3.2	Methods	5
1.3.3	Noise	5
1.4	The Recipe files	5
1.5	Chains' Association and Dissociation	5
1.6	Domain Reflection	6
1.6.1	Sphere	6

0.1 Simulation Framework

The beads- In our system of a chain of beads connected by harmonic springs, the beads represent monomers of a polymer chain.in our setting, these monomers can be thought of as the nucleosomes. Nucleosome- In Eukaryote cell, the nucleosome packs around 2 meters of DNA material into an accessible package called the nucleosome. It size is roughly $10\mu m$.

Chapter 1

Classes

In this chapter we review the different classes of the simulation framework and give details about the input/output of each method along with the properties of the class and concise explanation regarding the role of each of its methods. The sections' titles are brought here with the classes names as appearing in the code.

1.1 RouseSimulatorFramework

This class is the backbone of the simulation framework. It coordinates the action between all classes participating in the simulations. The class receives the parameters of each of its participating classes, distributes them and initializes each class.

1.1.1 Properties

- **handles** - holds the handles for classes and graphical object. All classes' handles are held under the fields *handles.classes*, all graphical handles, e.g figure, axes, buttons, et. are held under *handles.graphical*.
- **params** - holds the parameters for each of the participating classes. The parameters are parsed at the classes initialization from an .xml file (see section) and arranged as a structure placed in this property.

1.1.2 Methods

- **PreRunActions**- Actions performed before a simulation round begins.
- **PostRunActions** activates a sequence of predefined commands after each simulation round. In the present release the post run action is predefined to record simulation end time using *SimulatorDataRecorder.SetsimulationEndTime* method, save results using *SimulatorDataRecorder.SaveResults* method and notify about the simulation successful ending by email using *Send-Mail* function, located in the 3rdParty folder.

1.2 SimulationDataRecorder

Handles recording of the simulation data.

1.2.1 Properties

- **simulationData** Structure containing the fields:
 1. **chainObj** the Rouse chain object initialized using the Rouse class(see section 1.3)
 2. **numChains** number of chains.
 3. **step** current simulation step.
 4. **time** current simulation time, calculated as $step \times \Delta t$
 5. **positions**- the current position of the beads in the chain
 6. **beadDist** pair-wise bead distance matrix. This is a multidimensional matrix of size $[numBead \times numBead \times step]$
- **simulationRound** the current round of simulation
- **params** contains the parameters for the class.

paramList

- **saveType**- [all/external/none] (see SaveResults method)
- **resultsFolder** the path to the results folder. Can be relative or absolute

1.2.2 Methods

- **SaveResults** Implements 4 types of saving: [all/external/internal/none]
 1. *all*- save all data. Data is stored on the class and exported
 2. *external*- save all data to .mat files. No data is saved on the class
 3. *internal*- data is saved only on the class
 4. *none* - don't save any data. No data is saved on the class and none is exported to .mat files
- **ClearCurrentSimulationData** Clears the data from the class properties

1.3 Rouse

1.3.1 Properties

- **time**- time of the simulation, defined as $step \times \Delta t$
- **step**- simulation step
-
- **positions**
 1. **beads**- the coordinates of the beads relative to (0, 0, 0).
 - (a) **cur**- bead position at the present simulation step
 - (b) **prev**- bead position at the previous simulation step
 2. **springs**- The vectors defining the springs.
 - (a) **angleBetweenSprings**- this is a sparse representation of a 3D matrix defining the angle between bead i , j , and k . The convention is the position (i, j, k) represent the angle between bead i , j , and k , where i is the row, j is the column, and k is the depth (height) of the matrix. Springs are defined by a vector composed of subtracting the bead position i from $i + 1$.
 - (b) **length**- the length of the springs is the norm of each of the vectors defining the springs

1.3.2 Methods

- `setBeadsMobilityMatrix`
- `GetNewBeadsPosition`

1.3.3 Noise

The noise terms can be determined to be any type of noise distribution, with mean and variance as parameters. The default values are drawn from a Gaussian distribution with mean zero and variance 1. To save computation time for each step, the noise terms are determined every 10,000 simulation steps.

1.4 The Recipe files

The recipe files are used to allow the user to insert simulation specific commands without changing the content of the code. Recipe files allow the user to insert 4 functions, which are executed in: `PreSimulationBatchActions`, `PreRunActions`, `PostRunActions`, `PostSimulationBatchActions`. of the simulation framework.

1.5 Chains' Association and Dissociation

To explain the data structure's dynamics for chain association and dissociation, we first make several definitions.

A system of N chains will be written as: $[C^{(1)}, C^{(2)}, \dots, C^{(N)}]$, the coordinates of the beads in $C^{(i)}$ will be written as X^i . For each chain $C^{(i)}$ there exist a subset of 'sticky' beads $S^{(i)}$ that are allowed to interact with a subset $S^{(j)}$, $j \in [1, N]$. For the present notation we do not allow self interactions. A subscript will indicate a member of the group, i.e $X_k^{(i)}$ will indicate the k^{th} coordinate in the coordinates of chain i , $S_k^{(i)}$ will indicate the k^{th} coordinate in the subset of interacting beads of chain i , etc...

An *association* between a chain i and j is the interaction between a member of $S^{(i)}$ and $S^{(j)}$ and will be denoted by $C^{(i)} \oplus C^{(j)}$, that is $\exists \alpha \in S^{(i)}, \beta \in S^{(j)}; |X_\alpha^{(i)} - X_\beta^{(j)}| < \epsilon$, with ϵ the interaction distance.

A *dissociation* of the structure

1.6 Domain Reflection

1.6.1 Sphere

First we find the intersection point between the path at two consecutive time steps and the spherical domain. Let A be the particle location at time t and B the tentative location of the particle at time $t + \Delta t$, at any step we check if the path $C = B - A$ have crossed the spherical boundary by finding the roots of the quadratic equation for the intersection between a line and a sphere. setting $\alpha = \langle A, B \rangle - \langle A, A \rangle$, $\beta = \langle B - A, B - A \rangle$

$$t_{1,2} = \frac{-\alpha \pm \sqrt{\alpha^2 - \beta(\langle A, A \rangle - R^2)}}{\beta}$$

then we take the t such that $0 < t < 1$, to make sure the intersection is in the right direction and is between points A and B .