COMPUTATIONAL MODELING OF POLYMER CROWDING: INFLUENCE OF SOLVENT QUALITY AND DIMENSIONALITY ON CONFORMATIONS

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Wyatt Julian Davis

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Physics

April 2018

Fargo, North Dakota

NORTH DAKOTA STATE UNIVERSITY

Graduate School

Title

COMPUTATIONAL MODELING OF POLYMER CROWDING: INFLUENCE OF SOLVENT QUALITY AND DIMENSIONALITY ON CONFORMATIONS

By

Wyatt Julian Davis

ABSTRACT

The structure and function of polymers in confined environments, e.g., biopolymers in the cytoplasm, are affected by macromolecular crowding. To explore the influence of solvent quality and dimensionality on conformations of crowded polymers, polymers are modeled as penetrable ellipsoids/ellipses, whose shapes are governed by the statistics of random walks. Within this coarse-grained model, Monte Carlo simulations of two and three-dimensional polymer-nanoparticle mixtures, including trial changes in polymer size and shape, are performed. Penetration of polymers by nanoparticles is incorporated via a free energy cost predicted by polymer field theory. Simulation results of polymer conformation are compared with predictions of free-volume/area theory for polymers in good and theta solvents. Results indicate that dimensionality and solvent quality significantly affect crowded conformation, especially in the limit of small crowders. This approach may help to motivate future experimental studies of polymers in crowded environments, with relevance for drug delivery.

ACKNOWLEDGEMENTS

I'd like to first thank my ever supporting family. A warm thanks goes out to my girfriend, Shayna, and my friends in the Fargo-Moorhead swing dance community. Without you all I would have gone insane.

I thank my supervisor, Dr. Alan Denton, for all of the guidance and support he has given me.

Thanks to North Dakota State University for providing me with funds to travel to conferences and workshops that stimulated my intellectual development.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Macromolecular crowding occurs when the space available to a macromolecule in solution is reduced by the presence of other macromolecules or boundaries. Over the last several decades, this phenomenon has attracted strong interest within the biophysical community due to its ubiquity and influence in cellular and other biological environments [4]. The cellular interior is known to contain a dense population of macromolecules, occupying ∼20-40% of the available volume [5]. The resulting excluded volume effects alter the diffusional [6] and conformational behavior of larger biopolymers.

Crowded constraints in the cellular interior significantly modify biopolymer processes, such as protein folding, therein [7]. The impact of this crowded environment on polymer behavior has been investigated over the past several decades through various theoretical/computational, and experimental studies. In a computational study, the functionality of human telomerase RNA was found to be influenced by crowded conditions [8]. The results of a recent experimental study [9] indicate that crowding influences actin folding reactions. In an experimental study [1] DNA was subjected to crowding by dextran to replicate the crowded conditions inside the cytoplasm, it was found that not only the conformation responded but that DNA was well suited to diffuse easily though the crowded medium (see Fig. 1.1 for experimental images). Excluded volume interactions between macromolecules likely play a role in biopolymer organization within cellular interiors [10]. Macromolecular crowding has also been indicated as part of the pathogenesis of neurodegenerative diseases such as Alzheimer's [5].

Understanding how confinement of polymers in quasi-two dimensional environments like cell membranes impacts polymer conformation is key to gaining insight on their behavior in such environments. Cellular membranes are comprised of a lipid bilayer and embedded proteins. It is likely that the high areal fractions (30-55% [11]) of proteins in this assembly alter the free energies associated with protein (e.g. ion channel) conformations via a crowding effect [11]. In a recent experimental study [2], DNA was absorbed onto a cationic lipid membrane and it was found that the density of lipids influenced the conformation of the DNA molecules (see figure 1.2 for diagram of experimental setup). This experimental study partly inspired this investigation.

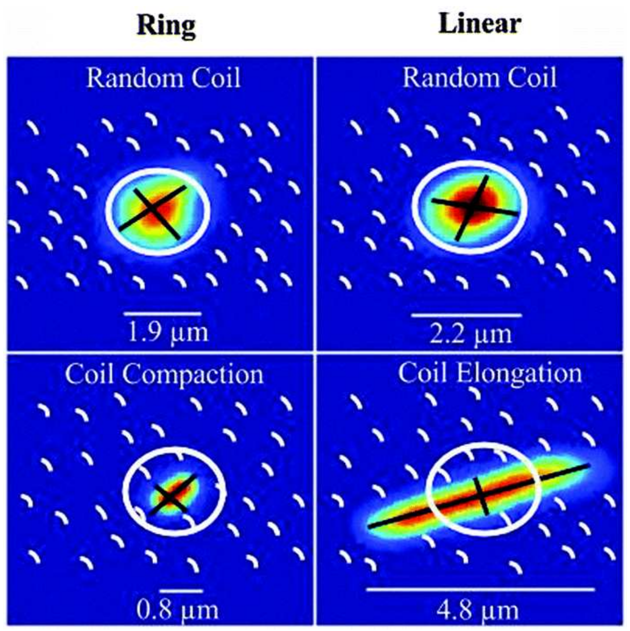

Figure 1.1. Fluorescently labeled DNA is shown to change conformation under the influence of crowding by dextran. Reproduced in part from [1] with permission of The Royal Society of Chemistry

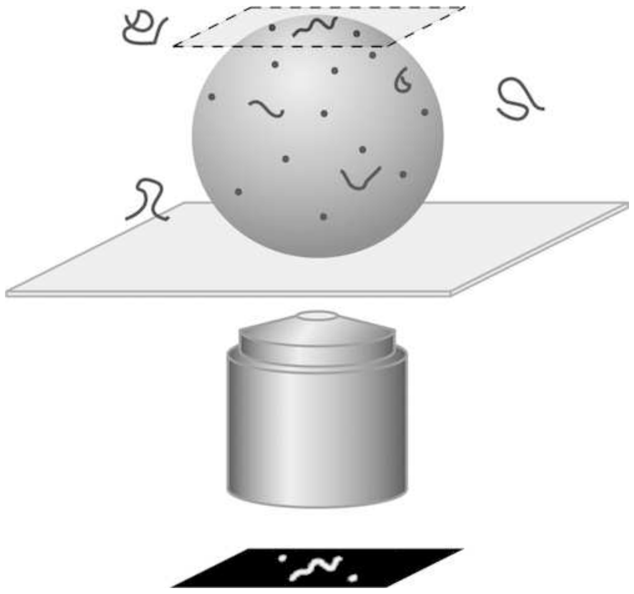

Figure 1.2. Experimental setup of fluorescently labeled DNA electrostatically bound to a freestanding fluid cationic lipid membrane. Conformations of DNA were observed to change with respect to the lipid headgroup concentration. Reproduced from [2] with permission of the American Physical Society.

It is well understood how solvent quality impacts the radius of gyration of linear homopolymers [12]. Solvent quality is a parameter that not only dictates polymer conformation [13], but also the thermodynamic behavior of polymer solutions [14], and the dielectric properties of polyelectrolyte solutions [15]. What is not as well understood is how solvent quality influences polymer size and shape in crowded environments.

Kuhn realized the aspherical nature of polymers when viewed from their principal axes frame, rather than the lab frame [16]. Mathematical studies of fluctuating random walk polymers have revealed that polymers are indeed aspherical and fluctuate in shape and size [17, 18, 19, 20, 21, 22]. By considering averages over random-walk shape polymers can be approximated as effective ellipsoids/ellipses. The coarse-grained ellipsoidal/elliptical model of a polymer used in these studies ties the conformation to random walk gyration-tensor eigenvalues. This study expands on previous works [3, 23, 24] to the analysis of the influence of solvent quality and dimensionality on the conformation of crowded polymers.

By comparing predictions from free-volume theory to results from molecular simulation, the importance of solvent quality, and solution dimensionality for the shape and size of crowded polymers is demonstrated. These parameters are probed using a simple coarse-grained model of hard nanosphere crowders plus ellipsoidal polymers in 3D and hard nanodisk crowders plus elliptical polymers in 2D.

In the next section, the details of the coarse-grained model are outlined. In the third section, the penetration model is discussed. Simulation methods and free volume theory are summarized in the fourth section. Finally, the theoretical and simulation results are discussed and conclusions are drawn in the last two sections.

2. MODELS

2.1. Coarse-Grained Model of a Polymer

2.1.1. 3D Model

A coarse-grained model of a polymer as a fluctuating ellipsoid is used to study the influence of crowding on polymer conformation. The coarse-grained approach used in this study is draws from previous works on crowding and depletion-induced effects [3, 23, 24].

The shape distribution of the polymer is governed by the gyration tensor of a random-walk:

$$\mathbf{T} \;=\; \frac{1}{N}\sum_{i=1}^{N} \mathbf{r}_i\; \mathbf{r}_i \;, \tag{2.1}$$

where $\mathbf{r}_i$ is the position relative to the center of mass of segment (step) i of N total segments. When viewed from the principal axis frame, the randomly walking coil takes on an average shape that can be characterized well by an ellipsoid. The surface of the ellipsoid can be expressed in terms of the gyration tensor eigenvalues

$$\frac{x^2}{\Lambda_1} + \frac{y^2}{\Lambda_2} + \frac{z^2}{\Lambda_3} = 3 \;, \tag{2.2}$$

where the first, second, and third eigenvalues correspond to the longest, second longest, and shortest principal axis of the ellipsoid. These eigenvalues determine the shape and size of the polymer.

The size of the polymer can be characterized by the radius of gyration:

$$R_{\mathrm{p}} = \left(\frac{1}{N}\sum_{i=1}^{N} r_i^2\right)^{1/2} = \sqrt{\Lambda_1 + \Lambda_2 + \Lambda_3} \;. \tag{2.3}$$

(the gyration tensor relates to the moment of inertia tensor $\mathbf{I}$ via $\mathbf{T} = R_p^2\mathbf{1} - \mathbf{I}$, with unit tensor $\mathbf{1}$.) The root-mean-square (rms) radius of gyration, which can be measured in scattering experiments, is given by

$$R_{\mathrm{g_{rms}}} = \sqrt{\langle R_{\mathrm{p}}^2\rangle} = \sqrt{\langle \Lambda_1 + \Lambda_2 + \Lambda_3\rangle} \;. \tag{2.4}$$

If the ensemble average in Eq. (2.4) is defined relative to a frame of reference that rotates with the polymer's principal axes and, furthermore, the principal axes are labelled to preserve the

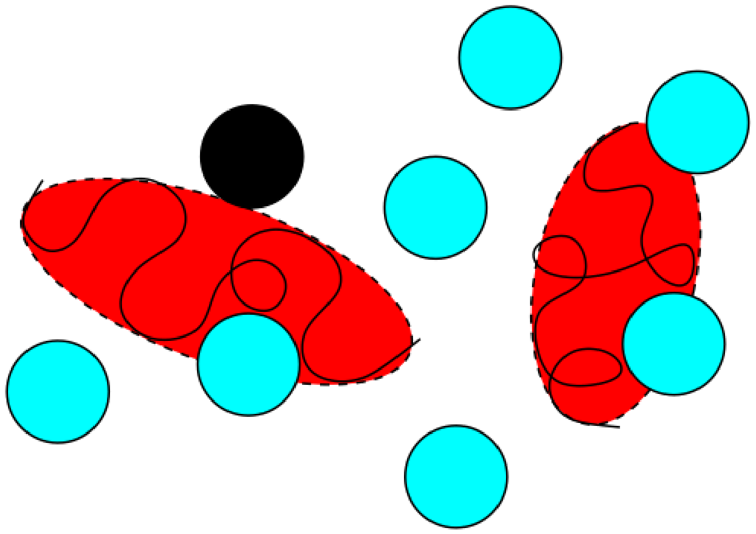

Figure 2.1. Model of a polymer crowded by nanoparticles. The polymer is represented as an ellipsoid that can fluctuate in size and shape and is penetrable by hard-sphere nanoparticles [3].

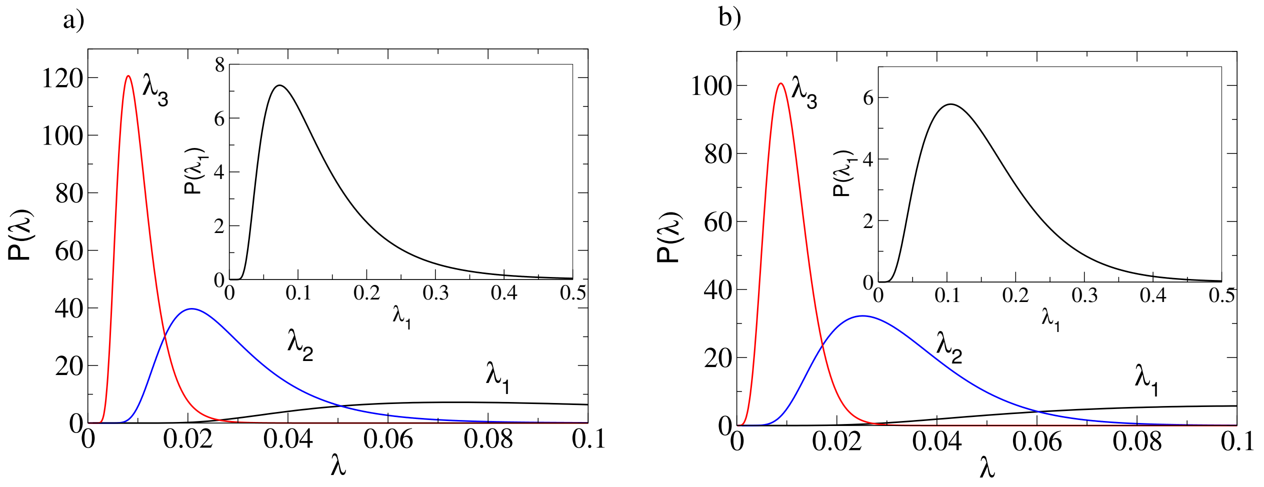

Figure 2.2. 3D polymer eigenvalue distributions for (a) random walk polymers and (b) self-avoiding walk polymers.

order of the eigenvalues from largest to smallest ($\Lambda_1 > \Lambda_2 > \Lambda_3$), then the average tensor describes an anisotropic object [25, 26].

A three-dimensional SAW, which models conformations of a linear, nonideal polymer in a good solvent, has an average shape determined by Monte Carlo simulations [27, 28] to be accurately described by a normalized probability distribution,

$$P_0(\Lambda_1, \Lambda_2, \Lambda_3) = \prod_{i=1}^{3} P_{i0}(\Lambda_i) \ , \tag{2.5}$$

where for a SAW polymer,

$$P_{i0}(\Lambda_i) = \frac{1}{\Gamma(\nu_i)} \frac{\nu_i}{\alpha_i} \left(\frac{\nu_i \Lambda_i}{\alpha_i} \right)^{\nu_i - 1} \exp\left(-\frac{\nu_i \Lambda_i}{\alpha_i} \right) \tag{2.6}$$

and α_i and ν_i are fitting parameters. The factorized form assumed in Eq. (2.5) implies independent eigenvalues. Although not exact, this assumption proves accurate, except for rare conformations in which an extreme extension in one direction affects the probability of an extension in an orthogonal direction. In the presence of nanosphere crowders, the shape distribution and the rms radius of gyration $R_g(\phi_c)$ depend on the volume fraction ϕ_c of the crowders. In the absence of crowders ($\phi_c = 0$), a SAW polymer of N segments, each of length l, has rms radius of gyration $R_g(0) = CN^{\nu}l$ with Flory exponent $\nu = 0.588$ and amplitude $C = 0.44108$ [28]. In a θ-solvent, $\nu = 0.5$, and $C = \frac{1}{\sqrt{6}}$ [28].Since the gyration tensor eigenvalues increase with N in proportion to $N^{2\nu}$, it is convenient to define scaled eigenvalues, $\lambda_i \equiv \Lambda_i/(N^{\nu}l)^2$. The shape distribution then can be expressed as

$$P_{i0}(\lambda_i) = a_i \lambda_i^{b_i} \exp(-c_i \lambda_i) \ , \tag{2.7}$$

where the fitting parameters a_i, b_i, and c_i, derived from α_i and ν_i, are given in Table 2.

For a RW polymer, the uncrowded eigenvalue probability distributions take the form

$$P_i(\lambda_i) = \frac{(a_i d_i)^{n_1 - 1} \lambda_i^{-n_i}}{2K_i} \exp\left(- \frac{\lambda_i}{a_i} - d_i^2 \frac{a_i}{\lambda_i} \right) \tag{2.8}$$

[29]. In terms of the scaled eigenvalues, the rms radius of gyration and the principal radii may be

Table 2.1. RW parameters for shape distribution in Eq. (2.8).

eigenvalue i	K_i	a_i	d_i	n_i
1	0.094551	0.08065	1.096	1/2
2	0.0144146	0.01813	1.998	5/2
3	0.0052767	0.006031	2.684	4

Table 2.2. SAW parameters for shape distribution in Eq. (2.7).

eigenvalue i	a_i	b_i	c_i
1	11847.9	2.35505	22.3563
2	1.11669×10^9	3.71698	148.715
3	1.06899×10^{14}	4.84822	543.619

expressed as

$$R_g(\phi_c) = \frac{R_g(0)}{C}\sqrt{\lambda_1 + \lambda_2 + \lambda_3} \tag{2.9}$$

and the principal radii may be expressed as

$$R_i(\phi_c) = \frac{R_g(0)}{C}\sqrt{3\lambda_i}\ . \tag{2.10}$$

For an ellipsoidal polymer with principal radii R_1, R_2, R_3, volume can be expressed exactly as

$$V = \frac{4\pi}{3}R_1R_2R_3 \tag{2.11}$$

or, in terms of scaled gyration tensor eigenvalues:

$$V(\phi) = \frac{4\pi}{3}\left(\frac{R_g(0)}{C}\right)^3\sqrt{27\lambda_1\lambda_2\lambda_3} \tag{2.12}$$

The deviation of a polymer's average shape from spherical can be quantified by an asphericity parameter [25, 26], defined as

$$A = 1 - 3\frac{\lambda_1\lambda_2 + \lambda_1\lambda_3 + \lambda_2\lambda_3}{(\lambda_1 + \lambda_2 + \lambda_3)^2}\ . \tag{2.13}$$

A perfect sphere has all eigenvalues equal and $A = 0$, while an elongated object with one eigenvalue much larger than the others has $A \simeq 1$. Crowding agents modify polymer size and shape.

As in previous studies of crowding of ideal polymers in a θ-solvent [3, 24, 23], this model extends the classic Asakura-Oosawa-Vrij (AOV) model of colloid-polymer mixtures [30, 31], which idealizes nonadsorbing polymers as effective spheres of fixed size (radius of gyration). Although qualitatively describing depletion-induced demixing of colloid-polymer mixtures, the AOV model completely neglects polymer conformational fluctuations and the influence of crowding on polymer size and shape. The AOV model also assumes that the polymers are impenetrable to the colloids.

2.1.2. 2D Model

The coarse-grained model for polymers confined to two dimensions is analogous to the model in 3D. The lowered dimensionality will not change the underlying scaling approach. Polymers are still modeled as random walks.

Within the 2D model, a polymer is treated as a coil of N identical connected segments. The coil has a size and shape characterized by it's gyration tensor T with components

$$T_{ki} = \frac{1}{N} \sum_{k=1}^{N} r_{ki} r_{kj} \tag{2.14}$$

where r_{ki} is the ith component of it's position vector r_k of the kth segment relative to the center of mass.

When viewed from the principal axis frame, the randomly walking coil takes on an average shape best characterized by an ellipse. Because of this, an ideal 2D random walk polymer coil can be modeled as a soft ellipse whose perimeter is expressed in cartesian coordinates as

$$\frac{x^2}{\Lambda_1} + \frac{y^2}{\Lambda_2} = 2 \tag{2.15}$$

where the first and second eigenvalues of this gyration tensor correspond to the longer and shorter principal axes of the ellipse representation of the polymer (which, in the case of equation 2.15, happen to lie on the x and y axes, respectively). The probability density function of an uncrowded conformation corresponding to dimensionless scaled eigenvalues

$$\lambda_i = \frac{\Lambda_i}{Nl^2}, \tag{2.16}$$

Table 2.3. Parameters for ideal (RW) shape distribution in Eq. (2.19).

eigenvalue i	a_i	b_i	c_i
1	139.2615	0.8769	13.5312
2	1.0176×10^6	2.1012	111.5845

Table 2.4. Parameters for non-ideal (SAW) shape distribution in Eq. (2.19).

eigenvalue i	a_i	b_i	c_i
1	4.8519×10^6	3.62054	49.3780
2	2.5501×10^8	2.75939	256.564

where l is the length of each individual segment, is given by

$$P_0(\lambda_1, \lambda_2) = P_1(\lambda_1)P_2(\lambda_2) \tag{2.17}$$

under the assumption that fluctuations of each axis are independent. The individual reservoir distributions of these eigenvalues are equivalent to

$$P_{i0}(\Lambda_i) = \frac{1}{\Gamma(\nu_i)}\frac{\nu_i}{\alpha_i}\left(\frac{\nu_i\Lambda_i}{\alpha_i}\right)^{\nu_i-1}\exp\left(-\frac{\nu_i\Lambda_i}{\alpha_i}\right). \tag{2.18}$$

This distribution can be recast in terms of the scaled eigenvalues using the form

$$P_{i0}(\lambda_i) = a_i\lambda_i^{b_i}\exp(-c_i\lambda_i) \tag{2.19}$$

The principal radii of the polymer can be expressed in terms of the scaled eigenvalues

$$\frac{R_i}{R_g^r} = \frac{\sqrt{2}}{C}\sqrt{\lambda_i} \tag{2.20}$$

where C=$\frac{1}{\sqrt{6}}$ for a RW polymer and C=.328862 for a SAW polymer. The size of the coil can be characterized by its radius of gyration

$$R_p = \left(\frac{1}{N}\sum_{i=1}^{N} r_i^2\right)^{\frac{1}{2}} = (\Lambda_1 + \Lambda_2)^{\frac{1}{2}} \tag{2.21}$$

which can be expressed, as above, in terms of the gyration tensor eigenvalues Λ_1 and Λ_2 (two

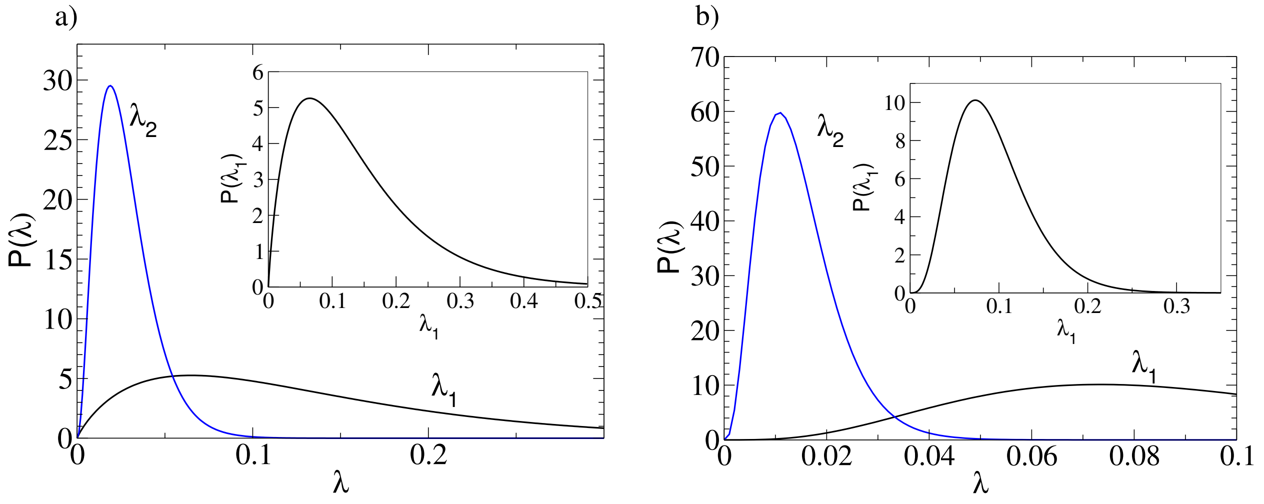

Figure 2.3. 2D polymer eigenvalue distributions for (a) random walk polymers and (b) self-avoiding walk polymers.

eigenvales for two dimensions). The root mean squared (rms) value of R_p

$$R_g = \sqrt{\langle R_p^2 \rangle} = \langle \Lambda_1 + \Lambda_2 \rangle^{\frac{1}{2}} \tag{2.22}$$

can be found experimentally. $\langle R_g \rangle$ can be expressed in terms of the scaled gyration tensor eigenvalues,

$$\langle R_g \rangle = \frac{1}{C} \langle \lambda_1 + \lambda_2 \rangle^{1/2} \tag{2.23}$$

The size of the polymer can also be characterized by the area

$$\langle \text{Area} \rangle = \pi \langle \mathrm{R}_1 \mathrm{R}_2 \rangle = \frac{2}{\mathrm{C}^2} \langle \sqrt{\lambda_1 \lambda_2} \rangle. \tag{2.24}$$

The shape of the polymer is characterized by the asphericity. A 2D polymer with an asphericity of 0 is a perfect circle, and a polymer with an asphericity of 1 is conformed as an elongated chain. asphericity can be expressed in terms of the scaled eigenvalues

$$\langle A \rangle = \frac{\langle (\lambda_1 - \lambda_2)^2 \rangle}{\langle (\lambda_1 + \lambda_2)^2 \rangle}. \tag{2.25}$$

2.2. Polymer-Nanoparticle Interaction

2.2.1. 3D Interaction

Penetration is allowed for with a penetration free-energy cost predicted by polymer-field theory and scaling theory. Following previous work [3, 23, 24, 32], the free energy cost of a spherical crowder of radius R_c penetrating the polymer, averaged over the surface of the polymer, is denoted as ε. The penetration free energy cost for a SAW (good solvent) polymer is predicted by polymer field theory [33, 34, 35, 36] as

$$\beta\varepsilon \propto \frac{1}{q^{1.29932}} \ , \tag{2.26}$$

where the polymer size ratio $q = R_g/R_c \gg 1$, $\beta = 1/(k_BT)$, and k_B is the Boltzmann constant. For a RW polymer, the free energy cost is predicted to be

$$\beta\varepsilon \propto \frac{1}{q}\left(1 + \frac{2}{q\sqrt{\pi}} + \frac{1}{3q^2}\right), \tag{2.27}$$

The dependence of $\beta\varepsilon$ on q can be justified by a scaling argument [12]. Assuming that $\beta\varepsilon$ is proportional to the fraction of polymer volume occupied by nanosphere in addition to the number of polymer segments(N), a scaling ansatz is made

$$\beta\varepsilon \sim (R_c/R_g)^3 Y(q) \ , \tag{2.28}$$

where $Y(q)$ is proportional to N. For a Flory exponent, ν, $N \sim R_g^{1/\nu}$. This implies that $Y(q) \sim q^{1/\nu}$, therefore

$$\beta\varepsilon \sim (R_c/R_g)^3 q^{1/\nu} \sim q^{1/\nu - 3} \ , \tag{2.29}$$

where ν=.5 for a θ-solvent and .588 for a good solvent.

The penetration-free energy profile is a step function. In the limit of small polymers ($q \approx 1$), the insertions of crowders becomes so energetically costly that polymers become virtually impenetrable and crowding occurs from outside of the polymer. In the limit of large polymers ($q \gg 1$) penetration is more likely to occur and crowding occurs from both the inside and outside of polymers.

2.2.2. 2D Interaction

Similarly to 3D, penetration of polymers is allowed in 2D following from polymer-field theory and scaling theory [12]. The penetration free energy cost for a disk penetrating a 2D SAW (good solvent) polymer is

$$\beta\varepsilon \propto \frac{1}{q^{2/3}} \qquad (q \gg 1), \tag{2.30}$$

and the penetration free energy for a RW (θ solvent) polymer is derived from polymer field theory:

$$\beta\varepsilon \propto 1 + \frac{4}{q\sqrt{\pi}} + \frac{1}{q^2}. \tag{2.31}$$

The dependence of ϵ on q can be justified by a simple scaling argument. Assuming that $\beta\varepsilon$ is proportional to the fraction of the polymer area occupied by the nanodisk and the number of polymer segments (N), a scaling ansatz is made

$$\beta\varepsilon \sim (R_c/R_p)^2 Y(q) \tag{2.32}$$

where $Y(q)$ is proportional to N. For a Flory exponent, ν, $N \sim R_g^{1/\nu}$. This implies that $Y(q) \sim q^{1/\nu}$, therefore

$$\beta\epsilon \sim (R_c/R_p)^2 q^{1/\nu} \sim q^{1/\nu - 2} \tag{2.33}$$

The penetration-free energy profile is a step function. In the limit of small q ($q \approx 1$) the polymer becomes virtually impenetrable and crowding is likely to occur from its outside. In the limit of large q ($q \gg 1$) the polymer becomes softer and susceptible to crowding from both its inside and outside.

3. COMPUTATIONAL METHODS

3.1. Monte Carlo Simulation

3.1.1. 3D Simulation

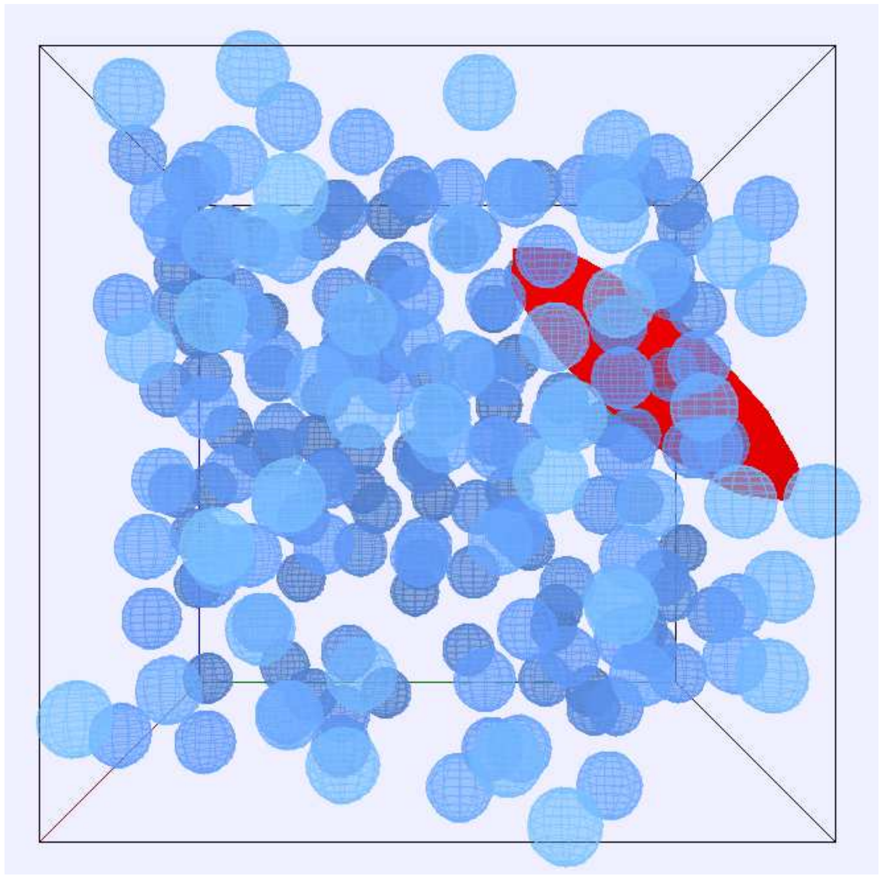

Figure 3.1. Snapshot of a simulation of $N_n = 216$ nanospheres (blue spheres) and one polymer (red ellipsoid) in a cubic simulation cell. The polymer rms radius of gyration in the reservoir equals five times the nanosphere radius (q=5).

Metropolis Monte Carlo (MC) simulation is used to calculate ensemble averages of crowded polymer geometric properties. The simulation is performed within a cubic cell with periodic boundary conditions. The volume of the cell, temperature, and number of particles in the cell are all fixed (canonical ensemble). Every trial move consists of a trial displacement of every nanosphere, a polymer displacement, polymer rotation, and polymer shape deformation. The tolerance for a nanosphere and polymer displacement is chosen to be .01 the nanosphere radius, the rotation tolerance is chosen to be .01 radians, and for the eigenvalues tolerances of $\Delta\lambda_1$ = .01, $\Delta\lambda_2$ = .003, $\Delta\lambda_3$ = .001 are used. The simulation was developed using the Open Source Physics Library in Java.

The acceptance probability for a trial change in configuration of the mixture is

$$P_{\text{config}}(\text{old} \rightarrow \text{new}) = \min\left\{ \frac{P_0(\lambda_{\text{new}})}{P_0(\lambda_{\text{old}})} e^{-\beta \Delta F}, 1 \right\} \tag{3.1}$$

where ΔF is the change in free energy associated with the configurational change, P_0 is the shape distribution of the uncrowded polymer, and λ represents a set of the three scaled gyration tensor eigenvalues. Hard spheres interact via a hard sphere potential, meaning no overlap is allowed. Consequently, any ΔF will be the result of a nanosphere overlapping or exiting the volume enclosed by a soft, ellipsoidal polymer. A move that creates/eliminates an overlap will result in a change in free energy equivalent to +/- $\beta\varepsilon$ (see section 2.2.1.). Within the simulation, determining whether a nanosphere overlaps a polymer involves calculating the roots of a 6th-order polynomial in order to find the closest point between a sphere and an ellipsoid.

By trial changes in the eigenvalues, and configuration of the hard-nanosphere fluid, the polymer is allowed to evolve towards a new ensemble average shape and size. See figure 3.1 for a snapshot of the simulation.

3.1.2. 2D Simulation

Metropolis Monte Carlo (MC) simulation is used to calculate ensemble averages of crowded polymer geometric properties. The simulation is performed within a square cell with periodic boundary conditions. The area of the cell, temperature, and number of particles in the cell are all fixed (canonical ensemble). Every trial move consists of a trial displacement of every nanodisk, a polymer displacement, polymer rotation, and polymer shape deformation. The tolerance for a nanodisk and polymer displacement is chosen to be .01 the nanodisk radius, the rotation tolerance is chosen to be .01 radians, and for the eigenvalues tolerances of $\Delta\lambda_1$ = .01, $\Delta\lambda_2$ = .001 are used. The simulation was developed using the Open Source Physics Library in Java.

The acceptance probability for a trial change in configuration of the mixture is the same as in eq. 3.1. where ΔF is the change in free energy associated with the configurational change, P_0 is the shape distribution of the uncrowded polymer, and λ represents a pair of the scaled gyration tensor eigenvalues. Hard disks interact via a hard-disk potential, meaning no overlap is allowed.

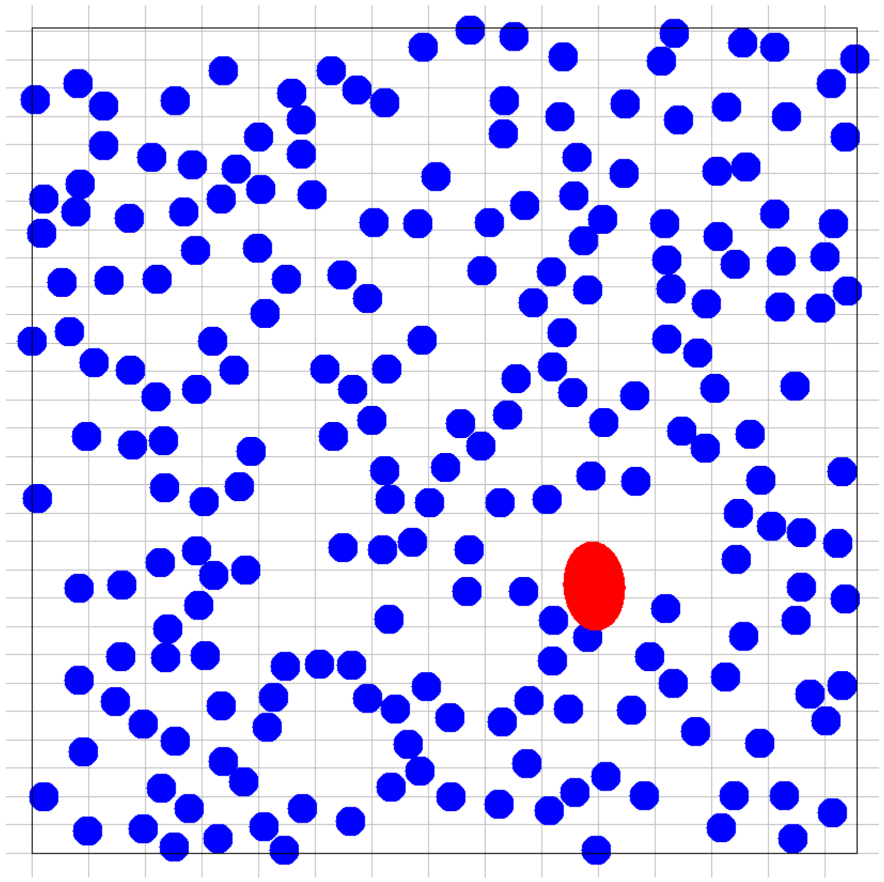

Figure 3.2. Snapshot of a simulation of $N_n = 216$ nanodisks (blue disks) and one polymer (red ellipse) in a square simulation cell. The polymer rms radius of gyration in the reservoir equals five times the nanodisk radius (q=5).

Consequently, any ΔF will be the result of a nanodisk overlapping or exiting the area enclosed by a soft, elliptical polymer. A move that creates/eliminates an overlap will result in a change in free-energy equivalent to +/- $\beta\varepsilon$ (see section 2.2.2.). Within the simulation, determining whether a nanodisk overlaps a polymer involves calculating the roots of a 4th-order polynomial in order to find the closest point between a disk and an ellipse.

By trial changes in the eigenvalues, and configuration of the hard-disk fluid, the polymer is allowed to evolve towards a new ensemble average shape and size. See figure 3.2 for a snapshot of the simulation.

3.2. Free-Volume Theory

3.2.1. 3D Theory

Free-volume theory of crowding is a generalization of the theory of Lekkerkerker *et al.* [37] from ensembles of hard, spherical polymers to soft, aspherical polymers. This mean-field theory is used to guide the choice of simulation parameters and for theoretical predictions to compare with simulation results of the model described in Sec. 2.1.1. and 2.2.1.. The following is a summary of the free-volume theory used in recent relevant studies [3, 24, 23].

The key value in the theory is the effective free volume fraction, $\alpha_{\text{eff}}(\phi_c)$, defined as the average free volume fraction available to a polymer amid a hard sphere fluid of volume fraction ϕ_c. The effective free volume fraction can be used to predict the crowded-polymer shape distribution

$$P(\lambda;\phi_c) = P_r(\lambda)\frac{\alpha(\lambda;\phi_c)}{\alpha_{\text{eff}}(\lambda;\phi_c)} \tag{3.2}$$

where λ is a set of scaled gyration tensor eigenvalues$(\lambda_1,\lambda_2,\lambda_3)$ characterizing a unique conformation of the polymer, $P_r(\lambda)$ is the uncrowded shape distribution of the polymer, and $\alpha(\lambda;\phi_c)$ is the free volume fraction available to a polymer with a conformation characterized by λ in a hard sphere fluid of volume fraction ϕ_c.

$\alpha_{\text{eff}}(\phi_n)$ is taken as an average of $\alpha(\lambda;\phi_c)$ over uncrowded polymer shapes

$$\alpha_{\text{eff}}(\phi_c) = \int d\lambda P_r(\lambda)\alpha(\lambda;\phi_c). \tag{3.3}$$

$\alpha(\lambda;\phi_c)$ is determined via generalized scaled-particle theory (see appendix for details on this calculation). P(λ_i), as described in sec. 2.1.1., is used to determine the shape distributions. The probability of each eigenvalue is determined by integrating over the other eigenvalues. As opposed to previous studies [24, 3, 23], a strict ordering of eigenvalues$(\lambda_3 < \lambda_2 < \lambda_1)$ is imposed. For example, consider finding the probability of some λ_1 value:

$$P_1(\lambda_1;\phi_c) = \int_{\lambda_3}^{\lambda_1} d\lambda_2 \int_0^{\lambda_2} d\lambda_3 P(\lambda;\phi_c). \tag{3.4}$$

To find the ensemble average of some crowded polymer geometric property B, it is taken as an integral over crowded polymer shape

$$\langle B\rangle = \int d\lambda B(\lambda)P(\lambda;\phi_c) \tag{3.5}$$

or, more explicitly

$$\langle B\rangle = \int_{\lambda_2}^{\infty} d\lambda_1 \int_{\lambda_3}^{\lambda_1} d\lambda_2 \int_{0}^{\lambda_2} d\lambda_3 B(\lambda_1,\lambda_2,\lambda_3)P(\lambda_1,\lambda_2,\lambda_3;\phi_c) \tag{3.6}$$

3.2.2. 2D Theory (Free-Area Theory)

Free-volume theory (outlined in the last section) can be scaled down to a free-area theory of crowding based on the same assumptions. This mean-field theory is used to guide choices of simulation parameters and for theoretical predictions to compare with simulation results of the model described in sec. 2.1.1 and 2.2.2.

Analogous to free-volume theory, the key value in free-area theory is the effective free-area fraction, $\alpha_{eff}(\phi_c)$, which is defined as the average free area fraction available to a polymer immersed in a hard-disk fluid of area fraction ϕ_c. The effective free-area fraction can be used to calculate the crowded shape distributions

$$P(\lambda;\phi_c) = P_r(\lambda)\frac{\alpha(\lambda;\phi_c)}{\alpha_{\text{eff}}(\lambda;\phi_c)} \tag{3.7}$$

where λ represents a unique pair of gyration tensor eigenvalues (λ_1,λ_2) that characterize the polymer conformation, $P_r(\lambda)$ is the uncrowded probability of that conformation, and $\alpha(\lambda;\phi_c)$ is the free-area fraction available to the polymer.

The effective free-area fraction is taken as an integral over polymer conformations while immersed in the hard disk fluid

$$\alpha_{\text{eff}}(\phi_c) = \int d\lambda P_r(\lambda)\alpha(\lambda;\phi_c). \tag{3.8}$$

$\alpha(\lambda;\phi_c)$ is calculated via generalized scaled particle theory (see the appendix for the details of this calculation). The probability for each scaled eigenvalue is determined by integrating over the other eigenvalue probabilities while maintaining a strict ordering of eigenvalues ($\lambda_2 < \lambda_1$). For example, when calculating the probability of λ_1 value:

$$P_1(\lambda_1; \phi_c) = \int_0^{\lambda_1} d\lambda_2 P(\lambda; \phi_c). \tag{3.9}$$

Ensemble average crowded polymer geometric properties (A) are taken as an integral over polymer conformation

$$\langle A \rangle = \int d\lambda A(\lambda) P(\lambda; \phi_c) \tag{3.10}$$

or, more explicitly

$$\langle A \rangle = \int_{\lambda_2}^{\infty} d\lambda_1 \int_0^{\lambda_1} d\lambda_2 A(\lambda_1, \lambda_2) P(\lambda_1, \lambda_2; \phi_c) \tag{3.11}$$

4. RESULTS

4.1. The Influence of Solvent Quality and Polymer Size on Conformation

Five independent MC simulations were run for each combination of crowder volume fraction and polymer size ratio (q). They consisted of N_n=216 nanoparticles initialized in a square lattice, and one soft ellipsoidal polymer (dilute polymer limit). Following 5×10^4 MC steps to reach equilibrium, the values of the three gyration tensor eigenvalues were collected 10^4 times at intervals of 10^3 MC steps. Using the eigenvalue data, ensemble averages of polymer gyration tensor eigenvalue distributions (Fig. 4.1), rms radius of gyration, effective volume, and asphericity (Fig. 4.2) are computed for SAW polymers. The error bars represent the standard deviation computed between the five independent runs.

Figure 4.1 shows simulation eigenvalue distribution data compared to free-volume theory for q=5 and q=10 respectively. Both simulation and theory display ensemble average eigenvalues shifting to smaller values, and a narrowing of distributions with increasing nanosphere volume fraction. The shifts in the distributions for q=10 are more substantial than those for q=5. This is clear from the changes in scale. These results indicate not only a tendency for the polymer to decrease in length along each axis, but also undergo smaller size fluctuations with increasing ϕ_c. Increasing deviation between simulation and mean field theory is expected with increasing ϕ_c as correlations between the polymer and nanospheres become more significant. The larger the polymer, the more severe the crowding effect on size and shape for a given ϕ.

Fig. 4.2, sub-figures a and b, show how polymer radius of gyration and volume (measures of size) change with respect to ϕ_c. Both radius of gyration and volume decrease with increasing ϕ_c. Polymers whose uncrowded size is the same as nanospheres (q=1) are relatively unaffected, experiencing a 15% decrease in radius of gyration at most. Polymers whose uncrowded sizes are substantially larger than nanospheres (q=5, and 10) experience a size change significantly larger than the smaller polymers at the same ϕ_c. The results are in good agreement with free-volume theory, with increasing deviation at higher ϕ_c and q where correlations are more severe. While the response of ideal polymers and nonideal polymers to crowding is similar in general trend, they differ quantitatively. Within the range of ϕ_c and q explored, larger nonideal polymers radius of

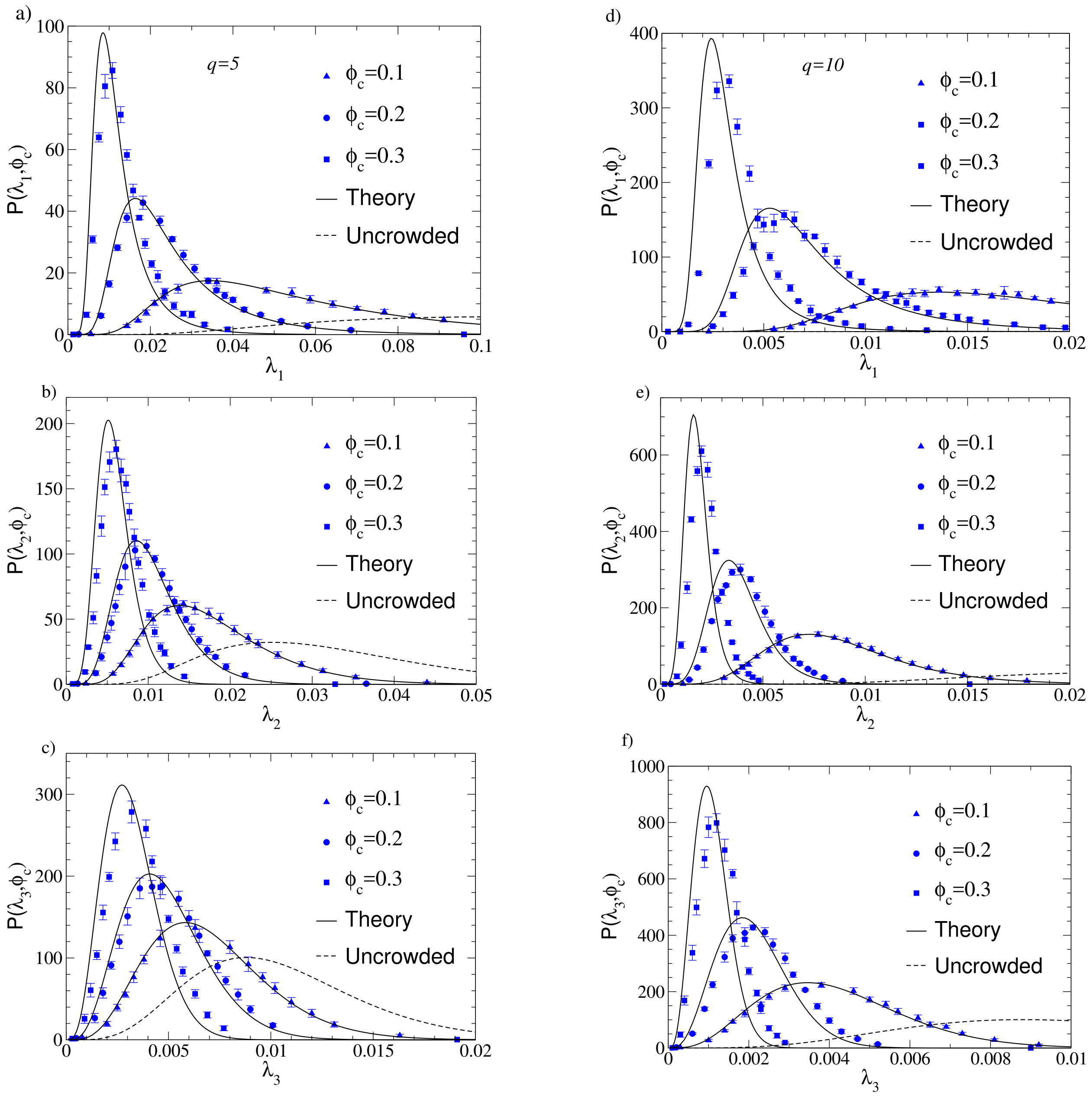

Figure 4.1. Probability distributions for the eigenvalues of the gyration tensor of a crowded polymer modeled as a self-avoiding walk: (a,d) λ_1, (b,e) λ_2, (c,f) λ_3. Simulation data (symbols) are compared with predictions of free-volume theory (solid curves) for a single ellipsoidal polymer, with uncrowded rms radius of gyration equal to 5 (a,b,c) and 10 (d,e,f) times the nanoparticle radius (q = 5,10), amidst N_n = 216 hard nanospheres with volume fraction ϕ_c = 0.1 (triangles), 0.2 (squares), and 0.3 (circles). Dashed curves show uncrowded distributions (ϕ_c = 0).

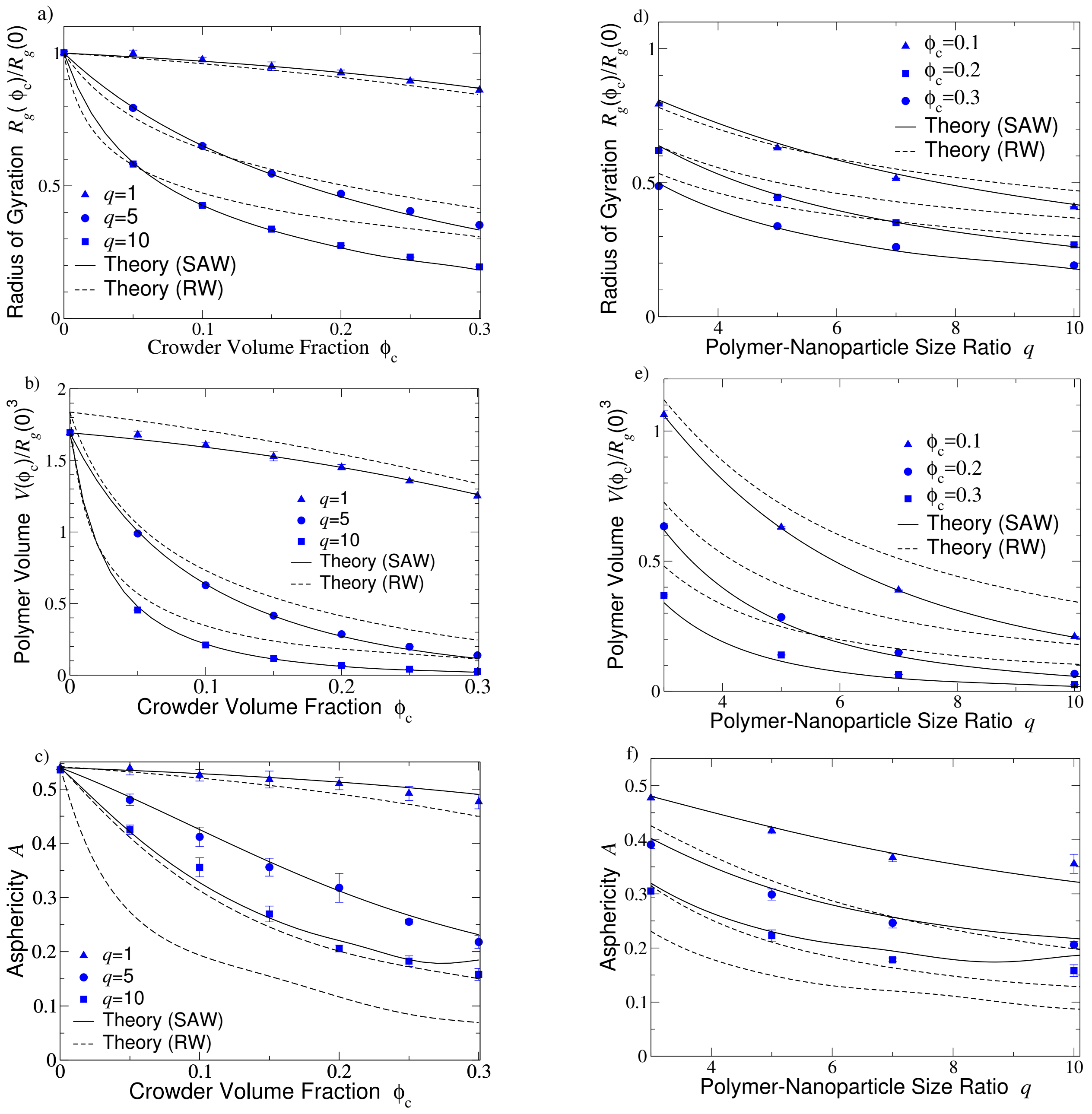

Figure 4.2. Geometric properties of polymers are shown. Some are with respect to ϕ and some are with respect to q. Simulation data are represented by points and free-volume theory is represented by solid/dashed curves.

gyration are less compressed at smaller ϕ_c when compared to their ideal counterparts, and are more compressed after some ϕ_c. The q=1 nonideal polymer is less compressed over the range of ϕ_c. These results indicate not only that the relative crowding impact on polymer size is dependent on the density of crowders, but also on the uncrowded size of the polymers.

To elucidate the influence of uncrowded polymer size on its crowded size, Fig. 4.2, subfigures d and f, show how polymer radius of gyration and volume change with respect to q for each ϕ_c. Both polymer radius of gyration and volume decrease with increasing q. At q=1, the difference in radius of gyration and volume values at different ϕ_c is small compared to the larger q values, consistent with the observations in the previous paragraph. As q gets larger the difference between radius of gyration values for different ϕ_c increases, indicating that larger polymer size responds more drastically to changes in crowder concentration.

Polymer shape was characterized with asphericity. In Fig. 4.2c, the asphericity for different q polymer with respect to crowder density ϕ_c is displayed. At the same q, compaction is more severe for the RW polymer than for the SAW polymer. RW polymer than it is for the SAW polymer. With increasing crowder volume fraction, both SAW and RW polymers become less aspherical, or more compact and spherical. At larger q, polymers compact more with increasing crowder volume fraction than at smaller q.

To make the relationship between uncrowded polymer size and crowded shape clearer, Fig. 4.2f. shows how polymer asphericity depends on the uncrowded size. The trend is similar for each constant ϕ_c value; the larger the polymer, the more its shape is deformed. At larger ϕ_c values, the more deformed the polymer; these results are consistent with the asphericity results displayed in figure 4.2c. So, not only does the asphericity of polymers depend on the crowder volume fraction, but also on the uncrowded size of the polymers. Furthermore, for similar values of ϕ_c and q, polymers in a good solvent are less crowded than those in a theta solvent, suggesting that they are more resilient to changes in shape under crowded conditions.

The observed influence of polymer size (q) on the crowding effect is consistent with the scaling relations outlined in section 2.2.1.. The volume of the polymer goes as q^3 ($V \propto q^3$). Using the SAW polymer as an example, the free energy cost of nanoparticle insertion goes as $\frac{1}{q^{1.29932}}$ $\left(\beta\varepsilon \propto \frac{1}{q^{1.29932}}\right)$. It follows that the overall energetic contribution of crowding goes as $q^{1.7}$ ($\Delta F \propto q^{1.7}$) indicating that the crowding effect increases significantly with q.

4.2. Constant Number of Segments

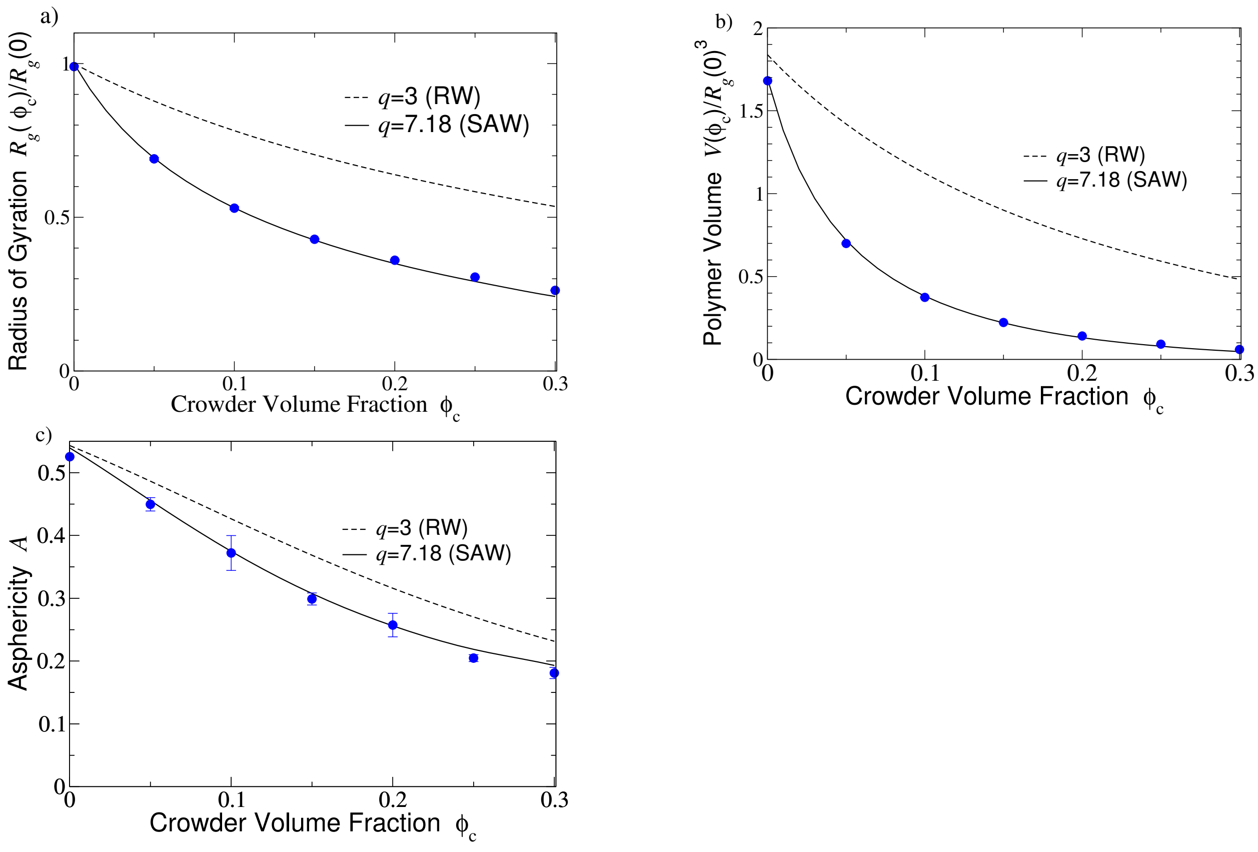

Figure 4.3. Predictions for PEG conformations crowded by Ficoll 70. Comparison is made between crowded SAW and RW PEG, each comprised of the same number of segments (N $\approx$ 6000).

Up until this point polymers in a theta solvent (RW) have been compared to those in good solvent (SAW) at fixed q. Polymer radii of gyration scale differently with number of segments depending on the solvent quality [12]. So, the comparison up until now has been between polymers of different numbers of segments in different solvent qualities. Using the scaling relations outlined in section 2.1.1 to vary q, the crowded conformation of a polymer with an equal number of segments is compared between a good and theta solvent.

Given the persistence length (segment length) of a linear polymer, the radius of a spherical crowder, and q for a polymer in a theta solvent, it is possible to calculate q for a polymer of equal segment length in a good solvent. A system of polyethylene glycol (PEG) and Ficoll 70 is considered. The persistence length of PEG in water at room temperature is 3.8 Å [38], and the radius of ficoll 70 is 55 Å [39]. q_θ is chosen to be 3, roughly the size of polymer used in a recent experimental crowding study [40]. For this choice of q_θ, $q_{good} = 7.18$ is obtained. Following from the scaling of q with N, this polymer is $\approx$ 6000 segments in length. Fig. 4.3 compares geometric properties between this crowded polymer in the presence of a good solvent and theta solvent.

As in the previous subsection, polymer size is characterized by the radius of gyration. As the crowder volume fraction is increased, the relative crowding effect on size on the polymer immersed in a good solvent is greater than that on the polymer immersed in a theta solvent. This is not surprising. This is consistent with the understanding that the crowding effect scales proportionately with q in general; q_{good} is over a factor of two larger than q_θ.

As in Sec. 4.1, the shape of the polymer is characterized with asphericity. As the crowder volume fraction is increased, the relative crowding effect on size on the polymer immersed in a good solvent is greater than that on the polymer immersed in a theta solvent. This is not surprising as the crowding effect on shape is proportional to q. What is interesting is how close the asphericities are. This good solvent resistance to shape change with crowder volume fraction is consistent with the results from the previous subsection.

Because we assume the polymer shape to be guided by equilibrium gyration tensor eigenvalue distributions, this model is only valid if the polymer has enough time to sample a reasonable number of microstates before a crowder diffuses too far. Let us assume that the polymer has time to equilibrate (visit a sizable region of its phase space) in the time it takes for one segment to diffuse a persistence length, and that to maintain equilibrium a spherical crowder shouldn't diffuse more than its own diameter in the same amount of time. This constraint puts a lower limit on the crowder diameter (of one persistence length) for which our equilibrium model is physically valid. In terms of our model, this translates to an upper limit of q dependant on the segment number and solvent quality. Consider the upper limit of q in the case of PEG (N$\approx$6000) in water at its theta temperature in water ($\approx$ 330 K [41]). Given $R_g(0) = l\left(\frac{N}{6}\right)^{\frac{1}{2}}$, we can conclude that the upper limit of q is 30. This upper limit of q puts the case of q=3 well within reasonable q values for this system.

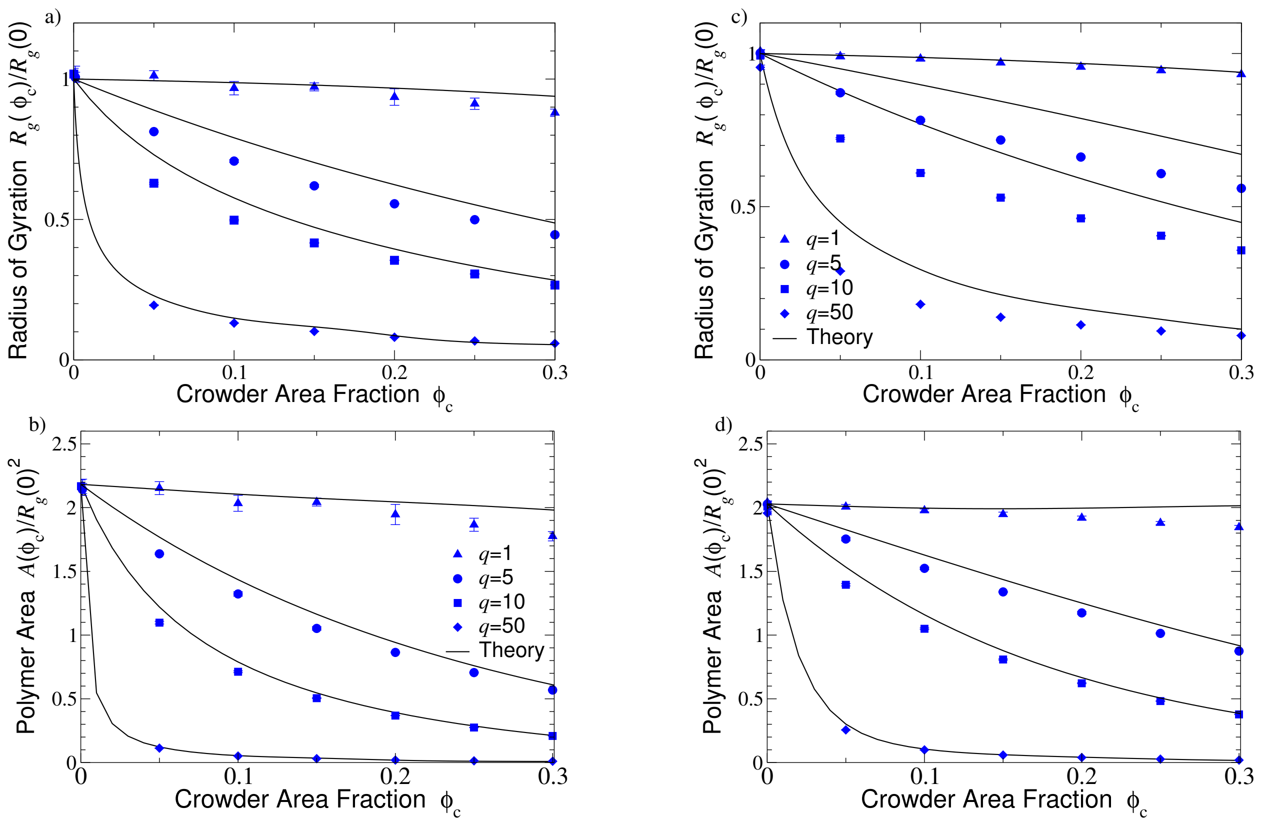

Figure 4.4. Geometric properties of 2D RW (a,b) and SAW (c,d) polymers are shown. The dots represent simulation data for q=1, 5, 10, 50 and solid curves represent theory.

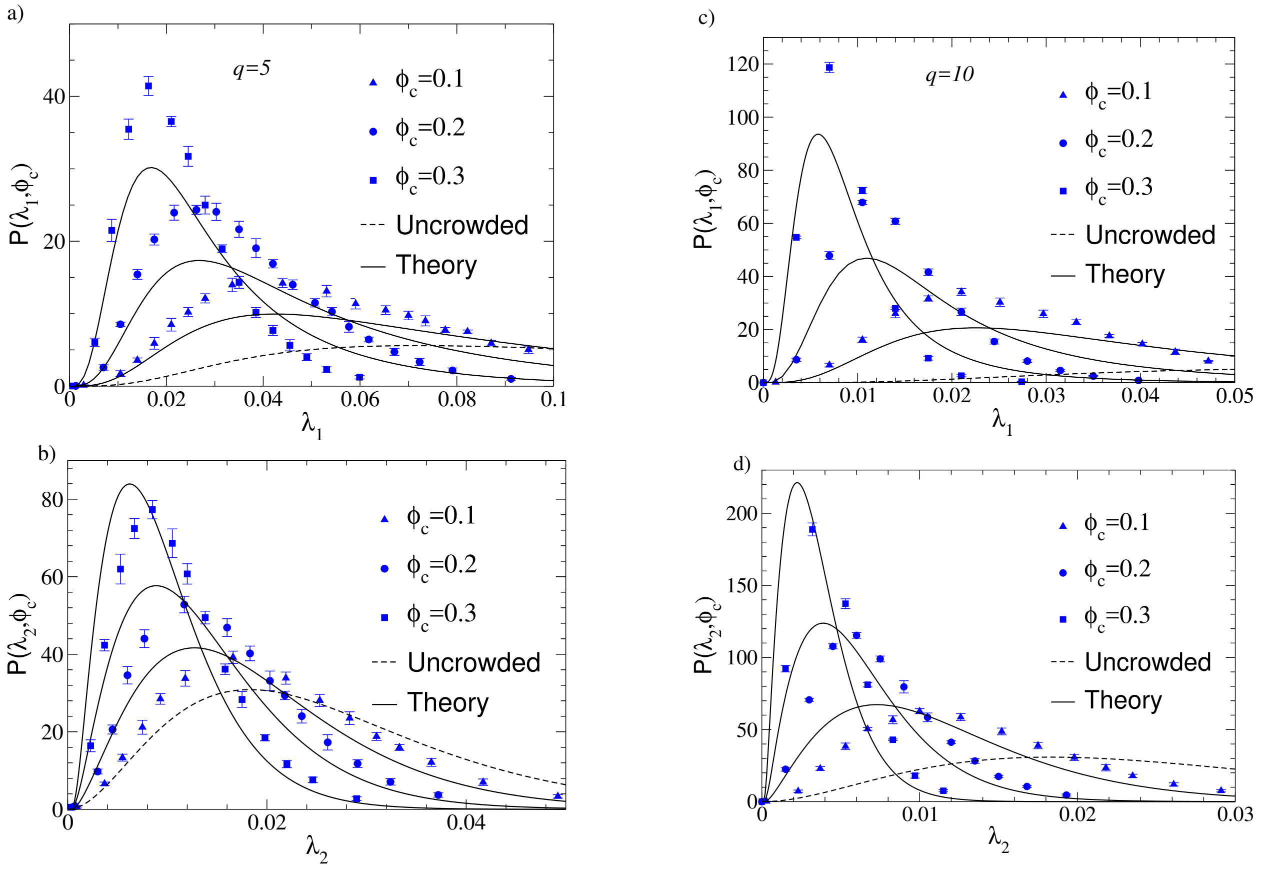

Figure 4.5. Eigenvalue distributions for crowded 2D RW polymers. The dashed curve represents the uncrowded distribution, the solid lines represent the free-area theory predictions, and the data points represent simulation data.

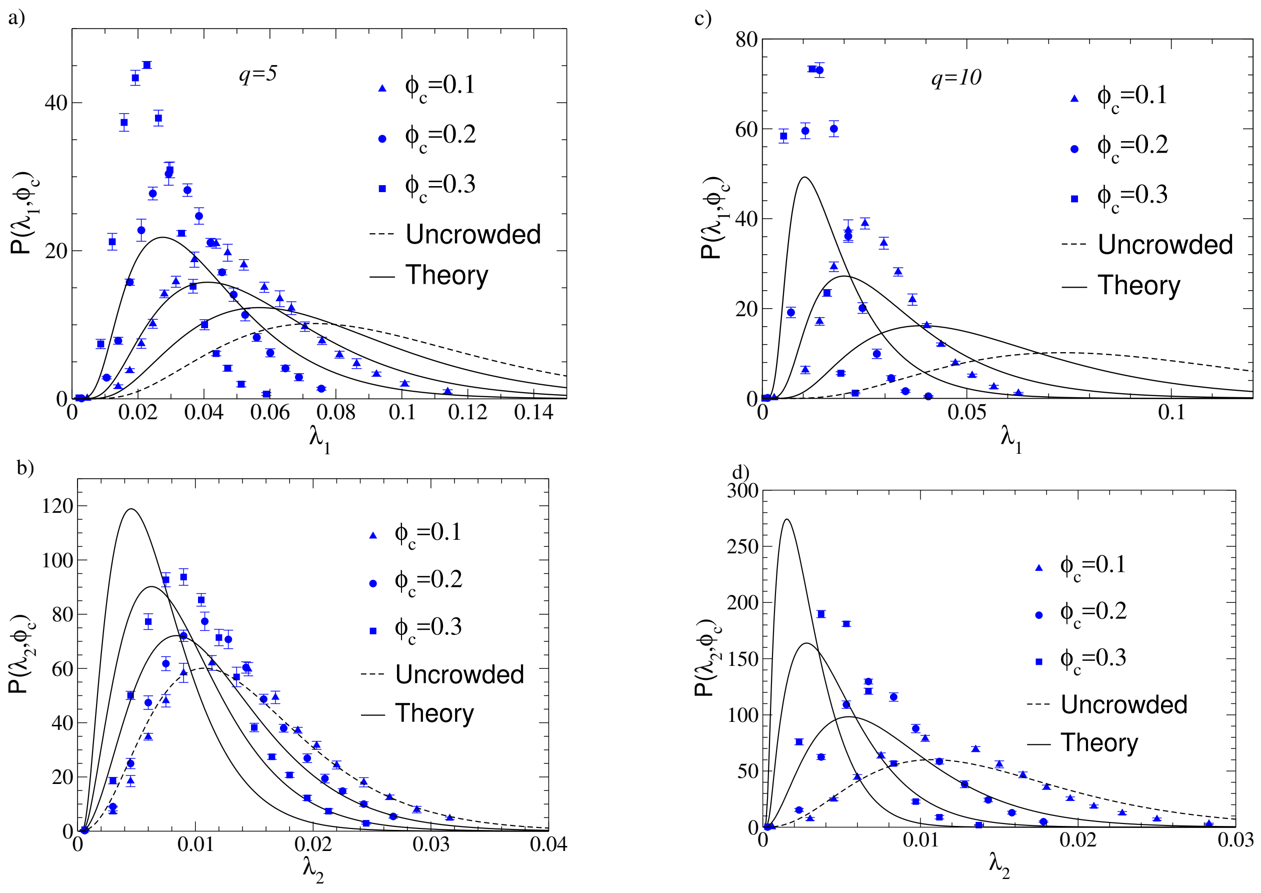

Figure 4.6. Eigenvalue distributions for crowded 2D SAW polymers. The dashed curve represents the uncrowded distribution, the solid lines represent the free-area theory predictions, and the data points represent simulation data.

4.3. Two-Dimensional Crowding

Five independent MC simulations were run for each combination of crowder area fraction and polymer size ratio (q). They consisted of N_n=216 nanodisks initialized in a hexagonal lattice, and one soft elliptical polymer (dilute polymer limit) initialized in a random interstitial location while not overlaping with any disks. Following 5×10^4 MC steps to reach equilibrium, the values of the three gyration tensor eigenvalues were collected 10^4 times at intervals of 10^3 MC steps. Using the eigenvalue data, the geometric properties of the polymer were determined. These geometric properties included area, and radius of gyration.

Figures 4.4 and 4.6 show the theoretical and simulation results for the crowded eigenvalue distributions for the random walk and self-avoiding walk polymers respectively. Similarly to 3D, both the theoretical and simulation distributions shift to smaller eigenvalues with increasing crowder area fraction. In addition, the distributions get less wide, indicating that polymers assume a smaller range of size when crowded more. The change in distributions is greater for the larger polymer (q=10), this is clear by the scale of the distributions when compared to (q=5). The discrepancy between the theoretical and simulation distributions is much larger in 2D than in 3D. $P(\lambda_1)$ is consistently underpredicted while $P(\lambda_2)$ is consistently overpredicted. Noteworthy also is the limited influence of crowding on SAW $q = 5$ $P(\lambda_2)$ at $\phi_c = 0.1$. The theory for this curve doesn't deviate much from the uncrowded curve, even more extreme is the simulation data which seems to exactly reproduce the uncrowded distribution. Currently, there is no explanation for this odd behavior of $P(\lambda_2)$ for that case. The likely culprits for these large discrepancies are outlined at the end of this section.

Figure 4.4 displays the radius of gyration results for polymers confined to two dimensions scaled to the uncrowded radius of gyration. Fig. 4.4 (a) displays results for the RW case and (c) displays the results for the SAW case. As with crowding in 3D, the radius of gyration decreases with increasing crowder concentration for both solvent qualities. The crowding affect on radius of gyration is relatively small for polymers of q=1, which decrease by less than 10% between $\phi_c = 0$ to $\phi = 0.3$. As the size q increases, so does the crowding affect, this is apparent for both RW and SAW polymers. For a given q, the SAW polymer is more resistant to changes in radius of gyration against crowding than its RW counterpart. This is a rather complex comparison, however, as for fixed q a RW polymer has more segments than a SAW polymer(assuming equal segment length). The free-area theory agrees well with simulation, especially at larger q.

Figure 4.4 displays the area results for polymers confined to two dimensions scaled to the radius of gyration squared. 4.4b displays results for the RW case and (d) displays the results for the SAW case. Like with the radius of gyration results, the area was observed to decrease with increasing crowder area fraction for RW and SAW polymers alike. The crowding affect on polymer area is small for polymers of $q = 1$, at least when compared with larger q. For a given q, just like with radius of gyration, the SAW polymer is more resistant to changes in area against crowding than its RW counterpart. Again, this comparison is difficult to make because, given that the segments lengths are the same, the RW of equal q to the SAW will have a larger number of segments. The free-area theory agrees well with simulation, especially at larger q.

The asphericity results are omitted for the 2D section. Due to large discrepancies between free volume theory and simulation results, the comparison was not deemed useful. There are a couple of potential culprits for this large discrepancy.

It is well known that the higher the spatial dimension, the less correlations influence results due to the larger configurational space. It follows that correlations would matter more in lower spatial dimensions. As with the radius of gyration and area results, the discrepancy between mean field theory and simulation is larger in 2D than it is in 3D. It may be the case that asphericity is more sensitive to these correlations than the area and radius of gyration, particularly because asphericity is the only geometric property measured that depends on differences between eigenvalues.

The thermodynamic properties of the hard-disk fluid are less well understood than the hard-sphere fluid. In the mean field theory, the approach to evaluate the interfacial tension was to expand in terms of the polymer curvature with coefficients dependent on the area fraction of the hard-disk fluid. It it not currently known at which term this expansion can be reasonably truncated. Monte Carlo simulations of hard disk fluids at hard curved interfaces were run to extract some of the coefficients with some limited success [42]. The theoretical asphericity exhibited high sensitivity to the values of these expansion coefficients, so even minor errors in coefficient values could result in significantly large errors for asphericity. See appendix for more details on this expansion of the interfacial tension for free-area theory.

5. CONCLUSIONS

To summarize, in this study the dependence of crowded polymer conformation on solvent quality and dimensionality was investigated. Taking advantage of the aspherical shape of random walks, polymers were modeled as effective ellipsoids. Polymers were allowed to fluctuate in shape and size according to random walk gyration tensor eigenvalue distributions. Hard nanoparticles interact mutually via a hard sphere potential, and are allowed to penetrate the volume enclosed by polymers with a free energy cost predicted by polymer field theory. Results from Monte Carlo simulations were compared with predictions from free-volume theory for ideal and nonideal polymers. Results indicate that polymers become smaller and more compact with increasing crowder volume fraction and uncrowded polymer size q, displaying good statistical agreement with free-volume theory. While there were similar qualitative trends in conformation between ideal and nonideal polymers, they differed quantitatively. When holding the number of segments constant, the crowding effects were stronger for polymers in a good solvent than in a theta solvent. This was likely a result of their swollen size and was especially evident for larger q. When the polymer-nanoparticle mixture was confined to two dimensions, the effects of crowding were qualitatively similar to 3D. The discrepancies between the free-volume theory and simulation results are larger in two dimensions. A likely contriubutor to these larger discrepancies is correlations, which are more pronounced in lower spatial dimensions. Another culprit could be an incomplete understanding of how the interfacial tension between the hard disk fluid and polymer depends on area fraction.

The model used in this study can be used and modified in order to explore specific polymers or other aspects of polymer-nanoparticle mixtures. It is possible to compute the shape distributions of individual biopolymers using molecular-scale simulation. These size distributions can then be mapped to the coarse-grained model and one could measure the effect of nanoparticle-crowding on the conformation of specific biopolymers. Using these simulation methods, one could probe the phase behavior of polymer-nanoparticle mixtures. Rather than using a simple step penetration free energy profile, one could use a profile that decreases radially, consistent with the monomer density profile of a real polymer. For comparison to the results of this paper, MD simulations could be run using a more explicit bead-spring polymer model. MD simulations could also be used to probe the dynamical properties of polymers under similar conditions, something the methods used in this study cannot do.

REFERENCES

[1] S. M. Gorczyca, C. D. Chapman, and R. M. Robertson-Anderson, "Universal scaling of crowding-induced dna mobility is coupled with topology-dependent molecular compaction and elongation," *Soft Matter*, vol. 11, pp. 7762–7768, 2015.

[2] C. Herold, P. Schwille, and E. P. Petrov, "Dna condensation at freestanding cationic lipid bilayers," *Phys. Rev. Lett.*, vol. 104, p. 148102, Apr 2010.

[3] W. K. Lim and A. R. Denton, "Depletion-induced forces and crowding in polymer-nanoparticle mixtures: Role of polymer shape fluctuations and penetrability," *J. Chem. Phys.*, vol. 144, pp. 024904–1–10, 2016.

[4] R. Ellis, "Macromolecular crowding: obvious but underappreciated," *Trends Biochem. Sci.*, vol. 26, no. 10, pp. 597 – 604, 2001.

[5] S. Mittal, R. K. Chowhan, and L. R. Singh, "Macromolecular crowding: Macromolecules friend or foe," *BBA-Gen. Subj.*, vol. 1850, no. 9, pp. 1822 – 1831, 2015.

[6] X.-W. Huang, Y. Peng, J.-H. Huang, and M.-B. Luo, "A study on the diffusivity of polymers in crowded environments with periodically distributed nanoparticles," *Phys. Chem. Chem. Phys.*, vol. 19, pp. 29975–29983, 2017.

[7] R. Ellis, "Macromolecular crowding: an important but neglected aspect of the intracellular environment," *Curr. Opin. Struc. Biol.*, vol. 11, no. 1, pp. 114 – 119, 2001.

[8] N. A. Denesyuk and D. Thirumalai, "Crowding promotes the switch from hairpin to pseudo-knot conformation in human telomerase rna," *J. Am. Chem. Soc.*, vol. 133, no. 31, pp. 11858–11861, 2011.

[9] I. A. Gagarskaia, O. I. Povarova, V. N. Uversky, I. M. Kuznetsova, and K. K. Turoverov, "The effects of crowding agents dextran-70k and peg-8k on actin structure and unfolding reaction," *J. Mol. Struct.*, vol. 1140, no. Supplement C, pp. 46 – 51, 2017. Molecular Spectroscopy and Molecular Structure 2016.

[10] C. Jeon, Y. Jung, and B.-Y. Ha, “Effects of molecular crowding and confinement on the spatial organization of a biopolymer,” *Soft Matter*, vol. 12, pp. 9436–9450, 2016.

[11] M. Lindén, P. Sens, and R. Phillips, “Entropic tension in crowded membranes,” *Plos. Comput. Biol.*, vol. 8, pp. 1–10, 03 2012.

[12] P. G. de Gennes, *Scaling Concepts in Polymer Physics*. Ithaca: Cornell, 1979.

[13] S. Gooßen, A. R. Brás, W. Pyckhout-Hintzen, A. Wischnewski, D. Richter, M. Rubinstein, J. Roovers, P. J. Lutz, Y. Jeong, T. Chang, and D. Vlassopoulos, “Influence of the solvent quality on ring polymer dimensions,” *Macromolecules*, vol. 48, no. 5, pp. 1598–1605, 2015.

[14] C.-E. Brunchi, S. Morariu, M. Cazacu, and M. Bercea, “Influence of the solvent quality on the thermodynamic behavior of polymethylphenylsiloxane solutions,” *Ind. Eng. Chem. Res.*, vol. 49, no. 24, pp. 12740–12746, 2010.

[15] F. Bordi, C. Cametti, T. Gili, S. Sennato, S. Zuzzi, S. Dou, and R. H. Colby, “Solvent quality influence on the dielectric properties of polyelectrolyte solutions: A scaling approach,” *Phys. Rev. E*, vol. 72, p. 031806, Sep 2005.

[16] W. Kuhn, “Gestalt fadenförmiger Moleküle in Lösungen (The Shape of Fibrous Molecules in Solutions),” *Kolloid-Zeitschrift*, vol. 68, pp. 2–15, 1934.

[17] B. E. Eichinger, “Heat capacity of dilute polymer solutions,” *J Chem. Phys.*, vol. 53, no. 2, pp. 561–566, 1970.

[18] M. Gordon, “Statistical mechanics of chain molecules,” *Brit. Polym. J.*, vol. 2, no. 5, pp. 302–303, 1970.

[19] M. Fixman and W. H. Stockmayer, “Polymer conformation and dynamics in solution,” *Annu. Rev. Phys. Chem.*, vol. 21, no. 1, pp. 407–428, 1970.

[20] D. N. Theodorou and U. W. Suter, “Shape of unperturbed linear polymers: polypropylene,” *Macromolecules*, vol. 18, no. 6, pp. 1206–1214, 1985.

[21] J. Rudnick, A. Beldjenna, and G. Gaspari, “The shapes of high-dimensional random walks,” *J. Phys. A-Math Gen.*, vol. 20, no. 4, p. 971, 1987.

[22] M. Bishop, J. H. R. Clarke, A. Rey, and J. J. Freire, “The shape of linear and star polymers with and without excluded volume,” *J. Chem. Phys.*, vol. 94, no. 5, pp. 4009–4011, 1991.

[23] W. K. Lim and A. R. Denton, “Influence of polymer shape on depletion potentials and crowding in colloid-polymer mixtures,” *Soft Matter*, vol. 12, pp. 2233–2492, 2016.

[24] W. K. Lim and A. R. Denton, “Polymer crowding and shape distributions in polymer-nanoparticle mixtures,” *J. Chem. Phys.*, vol. 141, pp. 114909–1–10, 2014.

[25] J. Rudnick and G. Gaspari, “The asphericity of random walks,” *J. Phys. A: Math. Gen.*, vol. 19, pp. L191–L193, 1986.

[26] J. Rudnick and G. Gaspari, “The shapes of random walks,” *Science*, vol. 237, pp. 384–389, 1987.

[27] S. J. Sciutto, “Study of the shape of random walks,” *J. Phys. A: Math. Gen.*, vol. 27, pp. 7015–7034, 1994.

[28] S. J. Sciutto, “The shape of self-avoiding walks,” *J. Phys. A: Math. Gen.*, vol. 29, pp. 5455–5473, 1996.

[29] F. Eurich, P. Maass, and J. Baschnagel, “Gaussian ellipsoid model for confined polymer systems,” *J. Chem. Phys.*, vol. 117, no. 9, pp. 4564–4577, 2002.

[30] S. Asakura and F. Oosawa, “Surface tension of high-polymer solutions,” *J. Chem. Phys.*, vol. 22, p. 1255, 1954.

[31] A. Vrij, “Polymers at interfaces and the interactions in colloidal dispersions,” *Pure & Appl. Chem.*, vol. 48, pp. 471–483, 1976.

[32] M. Schmidt and M. Fuchs, “Penetrability in model colloid–polymer mixtures,” *J. Chem. Phys.*, vol. 117, no. 13, pp. 6308–6312, 2002.

[33] E. Eisenriegler, A. Hanke, and S. Dietrich, “Polymers interacting with spherical and rodlike particles,” *Phys. Rev. E*, vol. 54, pp. 1134–115, 1996.

[34] A. Hanke, E. Eisenriegler, and S. Dietrich, “Polymer depletion effects near mesoscopic particles,” *Phys. Rev. E*, vol. 59, pp. 6853–6878, 1999.

[35] E. Eisenriegler, "Small mesoscopic particles in dilute and semidilute solutions of nonadsorbing polymers," *J. Chem. Phys.*, vol. 113, pp. 5091–5097, 2000.

[36] E. Eisenriegler, A. Bringer, and R. Maassen, "Polymer depletion interaction of small mesoscopic particles: Effects beyond leading order and anisotropic particles," *J. Chem. Phys.*, vol. 118, pp. 8093–8105, 2003.

[37] H. N. W. Lekkerkerker, W. C.-K. Poon, P. N. Pusey, A. Stroobants, and P. B. Warren, "Phase behaviour of colloid + polymer mixtures," *EPL (Europhys. Lett.)*, vol. 20, no. 6, p. 559, 1992.

[38] H. Lee, R. M. Venable, A. D. MacKerell, and R. W. Pastor, "Molecular dynamics studies of polyethylene oxide and polyethylene glycol: Hydrodynamic radius and shape anisotropy," *Biophys. J.*, vol. 95, no. 4, pp. 1590 – 1599, 2008.

[39] B. van den Berg, R. Wain, C. M. Dobson, and R. J. Ellis, "Macromolecular crowding perturbs protein refolding kinetics: implications for folding inside the cell," *EMBO J.*, vol. 19, no. 15, pp. 3870–3875, 2000.

[40] S. Palit, L. He, W. A. Hamilton, A. Yethiraj, and A. Yethiraj, "Combining diffusion nmr and small-angle neutron scattering enables precise measurements of polymer chain compression in a crowded environment," *Phys. Rev. Lett.*, vol. 118, p. 097801, Mar 2017.

[41] C. Özdemirdinç, G. Kibarer, and A. Güner, "Solubility profiles of poly(ethylene glycol)/solvent systems. ii. comparison of thermodynamic parameters from viscosity measurements," vol. 117, pp. 1100 – 1119, 03 2010.

[42] F. Smallenburg and H. Löwen, "Private communication," 2018.

[43] S. M. Oversteegen and R. Roth, "General methods for free-volume theory," *J. Chem. Phys.*, vol. 122, no. 21, p. 214502, 2005.

[44] R. Roth, "Fundamental measure theory for hard-sphere mixtures: a review," *J. Phys.: Cond. Mat.*, vol. 22, no. 6, p. 063102, 2010.

[45] J. HANSEN and I. McDONALD, “Preface to the third edition,” in *Theory of Simple Liquids (Third Edition)* (J.-P. Hansen, , and I. R. McDonald, eds.), pp. v –, Burlington: Academic Press, third edition ed., 2006.

[46] B. Widom, “Some topics in the theory of fluids,” *J. Chem. Phys.*, vol. 39, no. 11, pp. 2808–2812, 1963.

APPENDIX

A.1. Free-Volume Theory

In section 2.2.1 the free-volume theory of crowding is summarized. This appendix details a more rigorous calculation of the free volume fraction ($\alpha(\lambda;\phi_c)$) available to the polymer in a hard sphere fluid of volume fraction ϕ_c. The following derivation is based on previous works [3, 23, 24].

Following from the theory of Lekkerkerker et al. describing mixtures of colloids and spherical incompressible polymers, the Helmholtz free energy density, $f = f_{id} + f_{int}$, can be separated into the ideal gas contribution (f_{id}) and the excess interparticle contribution (f_{int}). The latter contribution can be further separated into the hard nanoparticle-nanoparticle contribution ($f_c(\phi_c)$) and the polymer-nanoparticle contribution (f_p). By way of a mean-field approximation, f_p can be equated to the free energy density of ideal polymers confined to a free volume. In practice, this approximation works well in the dilute polymer limit, especially for polymers in a good solvent where polymer segement-segment interaction is significant.

For the model including shape-fluctuating polymers, the free energy needs to be averaged over the shape degrees of freedom of polymers confined to the nanoparticle mixture. With the assumption that the polymer conformational entropy in the hard-nanoparticle mixture is the same as when it is uncrowded ($k_B \ln P_r(\lambda)$), f_p is expressed by

$$\beta f_p(\phi_c, \phi_p) = -n_p \int d\lambda P(\lambda;\phi_c) \ln(P_r(\lambda)\alpha(\lambda;\phi_c)) \tag{A.1}$$

where $P(\lambda;\phi_c)$ is the crowded probability for a polymer conformation characterized by a unique set of scaled gyration tensor eigenvalues ($\lambda \equiv (\lambda_1, \lambda_2, \lambda_3)$), amidst a hard-sphere fluid of volume fraction $\phi_c \equiv (4\pi/3)n_c R_c^3$. The total ideal-gas free energy density is given by

$$\beta f_{id}(\phi_c, \phi_p) = n_p \int d\lambda P(\lambda;\phi_c)\ln(P_r(\lambda)\alpha(\lambda;\phi_c)) - 1 + n_n(\ln(\phi_c) - 1) \tag{A.2}$$

where $\phi_p \equiv (4\pi/3)n_p(R_g^r)^2$ is the volume fraction occupied by the polymer in the system and R_g^r is the uncrowded polymer rms radius of gyration.

For a canonical ensemble of polymers and nanoparticles, the chemical potential of polymers of a given conformation within the system equals the chemical potential in the reservoir. Consequently,

$$n_p(\phi_c)P(\lambda;\phi_c) = n_p^r P_r(\lambda)\alpha(\lambda;\phi_n). \tag{A.3}$$

Integrating over polymer conformations (λ) while remembering the normalization of the distribution $P(\lambda;\phi_c)$ yields

$$n_p(\phi_c) = n_p^r P_r(\lambda)\alpha_{\text{eff}}(\phi_c) \tag{A.4}$$

where the effective free-volume fraction is defined as the average free-volume fraction taken as an integral over uncrowded polymer conformation

$$\alpha_{\text{eff}}(\phi_c) \equiv \int d\lambda P_r(\lambda)\alpha(\lambda;\phi_c). \tag{A.5}$$

In the limiting case of $\phi_c \to 0$, $\alpha(\lambda) \to 1$, therefore $\alpha_{\text{eff}}(\phi_c) \to 1$. The crowded probability shape distribution can be expressed as

$$P(\lambda;\phi_c) = P_r(\lambda)\frac{\alpha(\lambda;\phi_c)}{\alpha_{\text{eff}}(\phi_c)}. \tag{A.6}$$

The complete free energy density can now be constructed using the system configurational contribution (f_c), the nanosphere-nanosphere interaction contribution (f_{hs}), and the polymer-nanosphere interaction (f_p)

$$\begin{gathered}\beta f(\phi_c,\phi_p^r) = \beta(f_c + f_{hs} + f_p) = \\ n_n(\ln\phi_c - 1) + \beta f_{hs} + n_p^r\alpha_{\text{eff}}(\phi_n)(\ln\phi_p^r - 1)\end{gathered} \tag{A.7}$$

In order to calculate the free volume fraction, we utilize the geometry based approximation of Oversteegen and Roth [43] which generalizes scaled particle theory from hard spheres to arbitrary shapes. This generaliztion is done by way of fundamental-measures theory [44] to separate the thermodynamic properties of the crowders from the geometric properties of the depletant. This produces

$$\alpha(\lambda;\phi_c) = (1-\phi_c)\exp[-\beta(pv_p + \gamma a_p + \kappa c_p)] \tag{A.8}$$

where v_p, a_p, and c_p are the volume, surface area, and integrated mean curvature of an ellipsoidal polymer, and p, γ, and κ are the bulk pressure, surface tension at a hard planar wall, and bending rigidity of a hard-sphere fluid. For the ellipsoidal model of a polymer, a_p and c_p are numerically evaluated using the principal radii, while the volume can be evaluated exactly from the principal radii: $v_p = (4\pi/3)R_1R_2R_3$. The thermodynamic properties of the hard-nanosphere fluid are approximated by the Carnahan-Starling expressions [45]:

$$\begin{aligned}
\beta f_{hs} &= n_n \frac{\phi_c(4-3\phi_c)}{(1-\phi_c)^2} \\
\beta p &= \frac{3\phi_c}{4\pi R_n^3} \frac{1+\phi_c+\phi_c^2-\phi_c^3}{(1-\phi_c)^3} \\
\beta\gamma &= \frac{3}{4\pi R_n^3}\left[\frac{\phi_c(2-\phi_c)}{(1-\phi_c)^2} + \ln(1-\phi_c)\right] \\
\beta\kappa &= \frac{3\phi_c}{R_n(1-\phi_c)}
\end{aligned} \tag{A.9}$$

A.2. Free-Area Theory

Free-area theory was briefly discussed in section 2.2.2. This appendix lays free-area theory out in more detail. The effective free-area fraction is taken as an integral over polymer shapes amidst a hard-disk fluid of area fraction ϕ_c can be expressed as

$$\alpha_{\text{eff}}(\phi_c) = \int_{\lambda_2}^{\infty} d\lambda_1 \int_{0}^{\lambda_1} d\lambda_2 \alpha(\lambda_1, \lambda_2) \tag{A.10}$$

Following from the Widom particle insertion theorem [46], the free-area fraction can be expressed as

$$\alpha(\lambda; \phi_c) = \exp[-\beta W(\lambda; \phi_c)] \tag{A.11}$$

where W is the average work required to insert a polymer of shape λ into a sea of hard disks of area fraction ϕ_c. The average work takes the form of the Helfrich free energy required to distort the hard-disk fluid and create the area and surface needed to accommodate the polymer. W can be expressed as

$$W(\lambda; \phi_c) = p(\phi) a_p(\lambda) + \oint_C ds \gamma(\phi_c; s) \tag{A.12}$$

where p and γ are the pressure and interfacial tension respectively. The integral is taken over the closed curve C, where s is the elliptical perimiter parameter of the polymer. Similarly to the approach in 3D, the geometric properties of the polymer are conveniently separated from the thermodynamic properties of the hard-disk fluid.

Moving forward, it is assumed that the interfacial tension depends on the curvature of the interface between the polymer and hard-disk fluid and can be expanded in powers of the curvature $1/R$ (where R is the radius of curvature):

$$\gamma(\phi_c; R(s)) = \gamma_\infty(\phi_c) + c_1(\phi_c)\frac{\sigma}{R(s)} + c_2(\phi_c)\left(\frac{\sigma}{R(s)}\right)^2 + c_3(\phi_c)\left(\frac{\sigma}{R(s)}\right)^3 + ... \tag{A.13}$$

where γ_∞ is the interfacial tension of a hard disk fluid at a flat interface ($R \to \infty$), where $R(s)$ is the local radius of curvature at some point along the elliptial circumference. The coefficients $c_1(\phi_c)$, $c_2(\phi_c)$, and $c_3(\phi_c)$, are proportional to the Tolman length and hard-disk fluid bending rigidities.

Substituting A.12 into A.13 gives us the relation

$$W(\lambda;\phi_c) = p(\phi_c)a_p(\lambda) + \gamma_\infty(\phi_c)l_p(\lambda) + 2\pi\sigma c_1(\phi_c) + c_2(\phi_c)\kappa_2(\lambda) + c_3(\phi_c)\kappa_3(\lambda) + ... \tag{A.14}$$

where $l_p(\lambda)$ is the perimeter of the polymer. Using the Gauss-Bonnet theorem, the linear term in the expansion is evaluated exactly, with Euler characteristic of the ellipse $\chi = 1$, and

$$\begin{aligned} \kappa_2(\lambda) &\equiv \oint_C ds \left(\frac{\sigma}{R(s)}\right)^2 \\ \kappa_3(\lambda) &\equiv \oint_C ds \left(\frac{\sigma}{R(s)}\right)^3 \end{aligned} \tag{A.15}$$

are the integrated-squared and -cubed curvatures of the polymer. Using A.11 and A.14, the free-area fraction can be constructed as

$$\lim_{\lambda\to 0}[c_2(\phi_c)\kappa_2(\lambda) + c_3(\phi_c)\kappa_3(\lambda) + ...] = -k_BT\ln(1-\phi_c) - 2\pi\sigma c_1(\phi_c). \tag{A.16}$$

We assume that the LHS disappears in this limit, so we can solve for $c_1(\phi_c)$:

$$c_1(\phi_c) = -\frac{k_BT}{2\pi\sigma}\ln(1-\phi_c) \tag{A.17}$$

A.3. Disk-Ellipse Overlap Algorithm

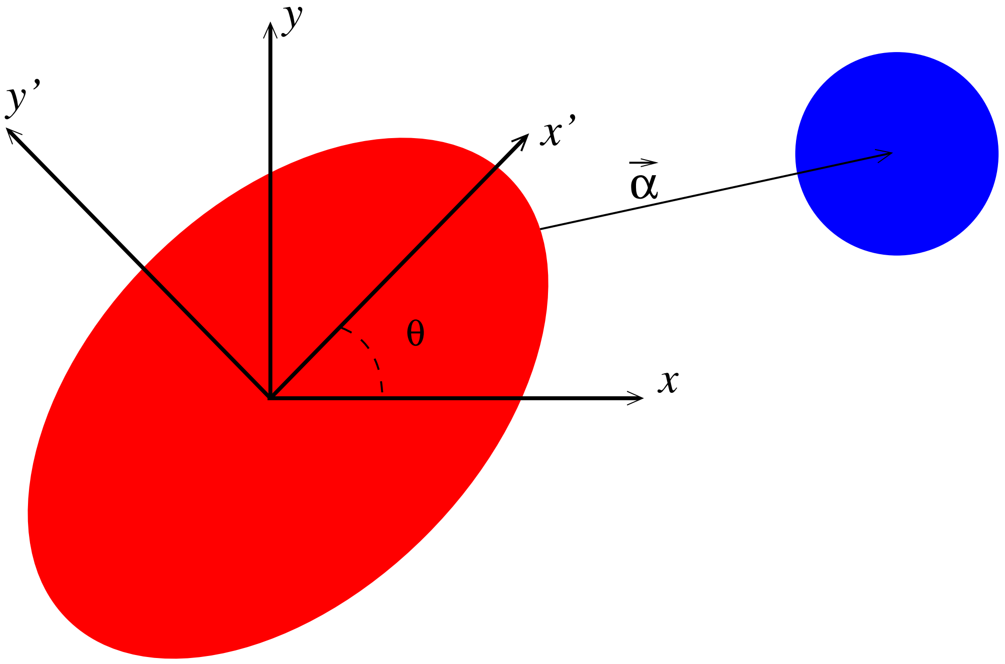

Figure 5.1. Overlap algorithm coordinate system and naming conventions.

To successfully simulate the crowding of elliptical polymers by hard disks, an accurate and efficient disk-ellipse overlap determination algorithm (ODA) was developed. For a given ellipse-disk pair, the goal of the ODA is to first determine the point along the circumference of the ellipse closest to the disk and then determine whether that disk is overlapping that point. The following appendix will cover the derivation and implementation of the ODA.

Consider a disk of radius d located at the cartesian coordinate $(x_0\ ,\ y_0)$ and an ellipse of principal radii a and b located at the origin where a and b lie on the x and y axis respectively. Consider θ, a rotation of the polymer following polar coordinate conventions. Using a standard two-dimensional rotation matrix

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{A.18}$$

the coordinates of the disk can be expressed in a new rotated coordinate system (x',y') via

$$\begin{bmatrix} x'_0 \\ y'_0 \end{bmatrix} = R(\theta) \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{A.19}$$

or,

$$\begin{aligned} x'_0 &= x_0 \cos\theta - y_0 \sin\theta \\ y'_0 &= -x_0 \sin\theta + y_0 \cos\theta. \end{aligned} \tag{A.20}$$

This convenient rotated reference frame, with the center of ellipse at the origin, will be used for the rest of the ODA derivation.

Any point along the perimeter of the ellipse whose tangent $d\vec{r} = \frac{dx'}{dy'}$ is perpendicular to the vector ($\vec{\alpha}$) pointing towards the disk from that point is one of two contenders for closest perimeter point to the disk, this condition is expressed as

$$\begin{aligned} \vec{\alpha} \cdot d\vec{r} = \alpha_x dx' + \alpha_y dy' = 0 \\ \Rightarrow \frac{dx'}{dy'} = -\frac{\alpha_y}{\alpha_x} \end{aligned} \tag{A.21}$$

We can express the tangent line in terms of the elliptical equation

$$\frac{dx'}{dy'} = -\frac{a^2}{b^2}\frac{y'}{x'} \tag{A.22}$$

and the components of $\vec{\alpha}$ as

$$\begin{aligned} \alpha_x &= x'_0 - x' \\ \alpha_y &= y'_0 - y'. \end{aligned} \tag{A.23}$$

Combining equations A.21, A.22, A.23, and utilizing the elliptical equation, we are left with a relation

$$\begin{aligned} -\frac{a^2}{b^2}\frac{y'}{x'} &= -\frac{y_0 - y}{x_0 - x} \\ \Rightarrow y' &= \frac{\frac{b^2}{a^2} y'_0 x'}{x'_0 + \left(\frac{b^2}{a^2} - 1\right) x'} \end{aligned} \tag{A.24}$$

or,

$$\frac{x'^2}{a^2} + \frac{\frac{b^2}{a^4} y_0'^2}{\left(\frac{x'_0}{x'} + \frac{b^2}{a^2} - 1\right)^2} = 1. \tag{A.25}$$

This condition can be re-cast as

$$\begin{gathered}\frac{A}{u^2}+\frac{B}{(u+c)^2}=1\\ \Rightarrow A(u+c)^2+Bu^2=u^2(u+c)^2\end{gathered} \tag{A.26}$$

where $u \equiv \frac{x_0}{x}$, $A \equiv \frac{x_0'^2}{a^2}$, $B \equiv \frac{b^2}{a^4}y_0'^2$, and $C \equiv \frac{b^2}{a^2}$.

Equation A.26 represents a forth-order polynomial equation in u that will yield two real and two imaginary roots. The two real roots are associated ($x' = \frac{x_0'}{u}$) with the two contenders for closest x' along the polymer's perimeter to the disk. The respective y' components can be extracted from the x' values using the elliptical equation. After solving for both coordinates, we can exactly determine two $\vec{\alpha}$. An overlap occurs if $|\vec{\alpha}| < d$.

Due to the non-trivial nature of root finding beyond 2nd-order polynomials, I opt for an eigenvalue algorithm to numerically determine the roots in the simulation. For the polynomial expressed in A.26, the respective companion matrix will take the form

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & -AC^2 \\ 1 & 0 & 0 & 0 & -2AC \\ 0 & 1 & 0 & 0 & C^2-B-A \\ 0 & 0 & 1 & 0 & 2C \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}. \tag{A.27}$$

The eigenvalues of the companion matrix are the roots of the fourth order polynomial. The eigenvalues are extracted with a numerical eigenvalue solver.

A.4. 2D Monte-Carlo Simulation Code

```
package org.opensourcephysics.sip.ch15;
import java.awt.*;
import java.awt.geom.Ellipse2D;
import java.awt.geom.AffineTransform;
import org.opensourcephysics.display.*;
import org.opensourcephysics.frames.*;
import org.opensourcephysics.numerics.*;
import java.util.stream.*;
import java.util.Arrays;


/**
 * DPM5App conducts a monte carlo simulation of a 2D polymer hard disk system based
    on user input.
 *
 *@author Wyatt Davis & Alan Denton based on MD code ch08/hd/HardDisksApp.java by
    Jan Tobochnik, Wolfgang Christian, Harvey Gould
 *@version 1.5 revised 04/18
 * Started as copy of DPM4.java
 */




public class DPM5 implements Drawable {
        public double x[], y[];
        public double xp[],yp[];
        public double a[],b[];
        public double l1[],l2[];
        public double theta[];
        public double sizeRatio;
        public double penetrationCost;
        public double al1 = 1.3690*10001/10000/Math.PI/Math.PI;
        public double al2 = 1.0972*10001/10000/2/2/Math.PI/Math.PI;
        public double v1 = 1.8769;
        public double v2 = 3.1012;
        public double b1 = 3.62051;
        public double b2 = 2.75939;
        public double c1 = 49.3780;
        public double c2 = 256.564;
        public int over[][];
        public int overtest[][];
        public double prob;
        public int Nd;
        public int Np;
        String solventQuality;
        public double phi;
        public double Lx,Ly;
        public double keSum = 0, virialSum = 0;
        public double steps = 0;
        public double tolerance;
        public double thetolerance = .05;
        public double l1tolerance = .005;
        public double l2tolerance = .001;
        public boolean condition = false;
        public boolean tworoots = true;
        public void initialize(String configuration) {
                resetAverages();
                x = new double[Nd];
                y = new double[Nd];
                xp = new double[Np];
                yp = new double[Np];
                a = new double[Np];
                b = new double[Np];
                l1 = new double[Np];
                l2 = new double[Np];
                over = new int[Np][Nd];
                overtest = new int[Np][Nd];
                theta = new double[Np];
                theta[0] = 0.;
                if(configuration.equals("regular")) {
                        setRegularPositions();
                }
                else {
                        setRandomPositions();
                }
        }


        public void resetAverages() {
                steps = 0;
                virialSum = 0;
        }

        public void setRandomPositions() {
                boolean overlap;
                for(int i = 0;i<Nd;++i) {
                        do {
                        overlap = false;
                        x[i] = Lx*Math.random();
                        y[i] = Ly*Math.random();
                        int j = 0;
                        while((j<i)&&!overlap) {
                                double dx = PBC.separation(x[i]-x[j], Lx);
                                double dy = PBC.separation(y[i]-y[j], Ly);
                                if(dx*dx+dy*dy<1.0) {
                                        overlap = true;
                                }
                        j++;
                        }
                        } while(overlap);
                }
        }

        //Set the initial positions of the disks and ellipse
        public void setRegularPositions() {
                double dnx = Math.sqrt(Nd);
                int nx = (int) dnx;
                if(dnx-nx>0.00001) {
                    nx++; // N is not a perfect square
                }
                double ax = Lx/nx; // distance between columns of disks
                double ay = Ly/nx; // distance between rows
                int i = 0;
                int iy = 0;
                while(i<Nd) {
                    for(int ix = 0;ix<nx;++ix) { // loops through disks in a row
                        if(i<Nd) {
                            y[i] = ay*(iy+0.5);
                            if(iy%2==0) {             // even rows
        displaced from odd rows
                                x[i] = ax*(ix+0.25);
                            }
                            else {
                                x[i] = ax*(ix+0.75);
                            }
                            i++;
                        }
                    }
                    iy++;
                }


                boolean overlap;
                for(int j = 0; j<Np; ++j){
                    do{
                        overlap = false;
                        xp[j] = Lx*Math.random();
                        yp[j] = Ly*Math.random();
                        a[j] = .5;
                        b[j] = .5;
                        if (solventQuality.equals("theta")){
                            l1[j] = Math.pow(a[j]/sizeRatio/.5,2)/12;
                            l2[j] = Math.pow(b[j]/sizeRatio/.5,2)/12;
                        }
```

```
                        else{
                                l1[j] = Math.pow(a[j]/sizeRatio/2.1506,2);
                                l2[j] = Math.pow(b[j]/sizeRatio/2.1506,2);
                        }
                        int k = 0;
                        while(overlap==false&&k<Nd){
                        double dx=PBC.separation(xp[j]-x[k], Lx);
                        double dy=PBC.separation(yp[j]-y[k], Ly);
                        if (dx*dx+dy*dy<(Math.pow(.5+.5,2))){
                                overlap = true;
                        }
                        k++;
                }
        } while(overlap);
        }

                                //Continue with random placement

}

//Probability distribution for theta-solvent
public void sciutto_th(int p, double dl1, double dl2){
        double L1O = 0; double L2O = 0;
        double L1N = 0; double L2N = 0;

        if(l1[p]>=l2[p]){
                L1O = l1[p]; L1N = L1O + dl1;
                L2O = l2[p]; L2N = L2O + dl2;

        }

        else if(l1[p]<l2[p]){
                L1O = l2[p]; L1N = L1O + dl2;
                L2O = l1[p]; L2N = L2O + dl1;
        }
```

```
            prob = Math.pow((L1N)/L1O,v1-1)*Math.pow((L2N)/L2O,v2-1)*Math.exp(-(
v1*(L1N-L1O)/al1)-(v2*(L2N-L2O)/al2));
    }

    //Probability distribution for good-solvent
    public void sciutto_go(int p, double dl1, double dl2){
            double L1O = 0; double L2O = 0;
            double L1N = 0; double L2N = 0;

            if(l1[p]>=l2[p]){
                    L1O = l1[p]; L1N = L1O + dl1;
                    L2O = l2[p]; L2N = L2O + dl2;
            }
            else if(l1[p]<l2[p]){
                    L1O = l2[p]; L1N = L1O + dl2;
                    L2O = l1[p]; L2N = L2O + dl1;
            }
            prob = Math.pow((L1N)/L1O,b1)*Math.pow((L2N)/L2O,b2)*Math.exp(-c1*(
L1N-L1O)-c2*(L2N-L2O));
    }

     //Make B a copy of array A (A and B must be of the same dimensions)
    public void arrayCopy(int A[][], int B[][]){
            for(int i=0; i<A.length; i++)
                    for(int j=0; j<A[i].length; j++)
                            B[i][j]=A[i][j];
    }

    //Determines whether overlap occurs between elliptical polymer and disk
    public void Overcond(int d, int p){
            // Step 1: Coordinate Transform
             tworoots = true;
             condition = false;
             double Txa = PBC.separation(x[d]-xp[p], Lx);
             double Tya = PBC.separation(y[d]-yp[p], Ly);
             double Tx = Math.cos(theta[p])*Txa-Math.sin(theta[p])*Tya;
```

```
          double Ty = Math.sin(theta[p])*Txa+Math.cos(theta[p])*Tya;
          // Step 2: Calcualte Coefficients of Polynomial
          double A = Tx*Tx/a[p]/a[p];
          double B = b[p]*b[p]*Ty*Ty/a[p]/a[p]/a[p]/a[p];
          double C = (b[p]*b[p]/a[p]/a[p]) - 1;
          // Step 3: Find Roots of Polynomial
         double coef[] = new double[5];
         coef[0] = A*C*C;
         coef[1] = 2*A*C;
         coef[2] = A+B-(C*C);
         coef[3] = -2*C;
         coef[4] = -1;

         double roots[][] = Root.polynomial(coef);
         double realroot1 = 0;
         double realroot2 = 0;

         // start of Alan's edits
          double realroots[] = new double[4];
          int nReal = 0;
          for(int i = 0; i < 4; i++){
                  if(roots[1][i]*roots[1][i] < 1.e-40){
                          realroots[nReal] = roots[0][i];
                          nReal++;
                  }
          }
          if(nReal != 2){
          tworoots = false;
          }
          else{
                  realroot1 = realroots[0];
                  realroot2 = realroots[1];
                  double xone = (Tx/realroot1);
                  double xtwo = (Tx/realroot2);
                  double yone = b[p]*b[p]*Ty*xone/a[p]/a[p]/(Tx+((b[p]*b[p]/a[
p]/a[p])-1)*xone);
```

```
                        double ytwo = b[p]*b[p]*Ty*xtwo/a[p]/a[p]/(Tx+((b[p]*b[p]/a[
p]/a[p])-1)*xtwo);
                        if((xone-Tx)*(xone-Tx)+(yone-Ty)*(yone-Ty) < .25){
                                condition = true;
                        }
                        else if((xtwo-Tx)*(xtwo-Tx)+(ytwo-Ty)*(ytwo-Ty) < .25){
                                condition = true;
                        }
                }
                // end of Alan's edits
        }



        //Conducts monte-carlo step of a system of hard-disks and soft elliptical
polymer
        public void step() {
                // make trial displacements and check for overlap
                tworoots = true;
                boolean overlap;
                double dxtrial, dytrial;
                double l1trial, l2trial;
                double L;
                double dtheta;
                for(int i = 0;i<Nd;++i) {
                        overlap = false;
                        tworoots = true;
                        dxtrial = tolerance*2.*(Math.random()-0.5);
                        dytrial = tolerance*2.*(Math.random()-0.5);
                        x[i] = PBC.position(x[i]+dxtrial, Lx);
                        y[i] = PBC.position(y[i]+dytrial, Ly);
                        for(int j = 0;j<Nd;++j) {
                                if((j!=i)&&!overlap) {
                                        double dx = PBC.separation(x[i]-x[j], Lx);
                                        double dy = PBC.separation(y[i]-y[j], Ly);
                                        if(dx*dx+dy*dy<1.0) {
                                                overlap = true;
```

```
                                            x[i] = PBC.position(x[i]-dxtrial, Lx
); // reject displacement
                                            y[i] = PBC.position(y[i]-dytrial, Ly
);
                                    }
                            }
                    }
                    condition = false;
                    tworoots = true;
                    for(int k = 0; k<Np; k++){
                            if(overlap != true && tworoots!=false){
                            L = a[k];
                            if(b[k]>=a[k]){
                                    L = b[k];
                            }
                            if(over[k][i]==1){
                                    Overcond(i,k);
                                    if(tworoots==true && condition==false){
                                            overtest[k][i] = 0;
                                    }
                            }
                            double Txa = PBC.separation(x[i]-xp[k],Lx);
                            double Tya = PBC.separation(y[i]-yp[k],Ly);
                            double dxp = Math.cos(theta[k])*Txa-Math.sin(theta[k
])*Tya;
                            double dyp = Math.sin(theta[k])*Txa+Math.cos(theta[k
])*Tya;
                            if(dxp*dxp + dyp*dyp<(L+.5)*(L+.5)){
                                    Overcond(i,k);
                                    if(tworoots==false){
                                            x[i] = PBC.position(x[i]-dxtrial, Lx
); // reject displacement
                                            y[i] = PBC.position(y[i]-dytrial, Ly
);
                                    }
                                    if(condition == true && tworoots==true){
```

```
                                                overtest[k][i] = 1;
                                        }
                                        if(condition == false && tworoots==true){
                                                overtest[k][i]=0;
                                        }
                                        if((dxp*dxp/a[k]/a[k])+(dyp*dyp/b[k]/b[k])
<1){
                                                overtest[k][i]=1;
                                        }


                                }
                        }
                }
                double rand = Math.random();
                if(tworoots){
                        if(rand<Math.exp(-penetrationCost*(IntStream.of(overtest[0])
.sum()-IntStream.of(over[0]).sum())) && tworoots==true){
                                arrayCopy(overtest,over);
                        }
                        else if(rand>Math.exp(-penetrationCost*(IntStream.of(
overtest[0]).sum()-IntStream.of(over[0]).sum())) && tworoots==true){
                                arrayCopy(over,overtest);
                                x[i] = PBC.position(x[i]-dxtrial, Lx); // reject
displacement
                                y[i] = PBC.position(y[i]-dytrial, Ly);
                        }
                }
                }
                condition = false;
                for(int k = 0; k<Np; k++){
                        condition = false;
                        tworoots = true;
                        dxtrial = tolerance*2.*(Math.random()-0.5);
                        dytrial = tolerance*2.*(Math.random()-0.5);
                        l1trial = l1tolerance*2.*(Math.random()-0.5);
                        l2trial = l2tolerance*2.*(Math.random()-0.5);
                        dtheta = thetolerance *2.*(Math.random() -0.5);
                        if (solventQuality.equals("theta")){
                                sciutto_th(k,l1trial,l2trial);
                        }
                        else{
                                sciutto_go(k,l1trial,l2trial);
                        }
                        l1[k] = l1[k]+l1trial;
                        l2[k] = l2[k]+l2trial;
                        if (l1[k]<0 || l2[k]<0){
                                condition = true;
                        }
                        if (solventQuality.equals("theta")){
                                a[k] = .5*sizeRatio*Math.pow(l1[k]*12,.5);
                                b[k] = .5*sizeRatio*Math.pow(l2[k]*12,.5);
                        }
                        else{
                                a[k] = 2.15016*sizeRatio*Math.pow(l1[k],.5);
                                b[k] = 2.15016*sizeRatio*Math.pow(l2[k],.5);
                        }
                        xp[k] = PBC.position(xp[k]+dxtrial,Lx);
                        yp[k] = PBC.position(yp[k]+dytrial,Ly);
                        theta[k] += dtheta;
                        if (condition == true){
                                l1[k] = l1[k]-l1trial;
                                l2[k] = l2[k]-l2trial;
                                if (solventQuality.equals("theta")){
                                        a[k] = .5*sizeRatio*Math.pow(l1[k]*12,.5);
                                        b[k] = .5*sizeRatio*Math.pow(l2[k]*12,.5);
                                }
                                else{
                                        a[k] = 2.15016*sizeRatio*Math.pow(l1[k],.5);
                                        b[k] = 2.15016*sizeRatio*Math.pow(l2[k],.5);
                                }
                                xp[k] = PBC.position(xp[k]-dxtrial, Lx); // reject
displacement
```

```
                        yp[k] = PBC.position(yp[k]-dytrial, Ly);
                        theta[k] -= dtheta;
                        continue;
                 }
                 L = a[k];
               if(b[k]>a[k]){
                        L = b[k];
               }
               for(int j = 0;j<Nd;++j){
                        if(tworoots==true){
                                 double Txa = PBC.separation(x[j]-xp[k],Lx);
                                 double Tya = PBC.separation(y[j]-yp[k],Ly);
                                 double dxp = Math.cos(theta[k])*Txa-Math.sin
(theta[k])*Tya;
                                 double dyp = Math.sin(theta[k])*Txa+Math.cos
(theta[k])*Tya;
                                 if(over[k][j]==1){
                                          Overcond(j,k);
                                          if(!tworoots){
                                                   continue;
                                          }
                                          if(tworoots==true){
                                                   if(condition==false){
                                                            overtest[k][j]=0;
                                                   }
                                                   if (condition==true){
                                                            overtest[k][j]=1;
                                                   }
                                          }
                                 }
                                 if(dxp*dxp+dyp*dyp<(L+.5)*(L+.5)){
                                          Overcond(j,k);
                                          if(tworoots==false){
                                                   continue;
                                          }
                                          if(condition==true){
                                                overtest[k][j]=1;
                                            }
                                            if(condition == false){
                                                overtest[k][j]=0;
                                            }
                                            if ((dxp*dxp/a[k]/a[k])+(dyp*dyp/b[k
]/b[k])<1){
                                                overtest[k][j]=1;
                                            }
                                        }
                                }
                        }

                        if (tworoots){
                                double rand = Math.random();
                                if(rand<prob*Math.exp(-penetrationCost*(IntStream.of(
overtest[0]).sum()-IntStream.of(over[0]).sum())) && tworoots==true){
                                        arrayCopy(overtest,over);
                                }
                                else if(rand>prob*Math.exp(-penetrationCost*(IntStream.of(
overtest[0]).sum()-IntStream.of(over[0]).sum())) && tworoots==true){
                                        l1[k] = l1[k]-l1trial;
                                        l2[k] = l2[k]-l2trial;
                                        arrayCopy(over,overtest);
                                        if (solventQuality.equals("theta")){
                                                a[k] = .5*sizeRatio*Math.pow(l1[k]*12,.5);
                                                b[k] = .5*sizeRatio*Math.pow(l2[k]*12,.5);
                                        }
                                        else{
                                                a[k] = 2.15016*sizeRatio*Math.pow(l1[k],.5);
                                                b[k] = 2.15016*sizeRatio*Math.pow(l2[k],.5);
                                        }
                                        xp[k] = PBC.position(xp[k]-dxtrial, Lx); //reject
displacement
                                        yp[k] = PBC.position(yp[k]-dytrial, Ly);
                                        theta[k] -= dtheta;
```

```
                    }
                }
                if (!tworoots){
                    l1[k] = l1[k]-l1trial;
                    l2[k] = l2[k]-l2trial;
                    arrayCopy(over,overtest);
                    if (solventQuality.equals("theta")){
                        a[k] = .5*sizeRatio*Math.pow(l1[k]*12,.5);
                        b[k] = .5*sizeRatio*Math.pow(l2[k]*12,.5);
                    }
                    else{
                        a[k] = 2.15016*sizeRatio*Math.pow(l1[k],.5);
                        b[k] = 2.15016*sizeRatio*Math.pow(l2[k],.5);
                    }
                    xp[k] = PBC.position(xp[k]-dxtrial, Lx); //reject
displacement
                    yp[k] = PBC.position(yp[k]-dytrial, Ly);
                    theta[k] -= dtheta;
                }

        }
        }

        //Draw the polymer-nanoparticle mixture
        public void draw(DrawingPanel drawingPanel, Graphics g) {
                double radius = 0.5;
                int pxRadiusP[] = new int[Np];
                int pyRadiusP[] = new int[Np];
                Graphics2D g2 = (Graphics2D) g; //New
                if(x==null) {
                    return;
                }
                int pxRadius = Math.abs(drawingPanel.xToPix(radius)-drawingPanel.
xToPix(0));
                int pyRadius = Math.abs(drawingPanel.yToPix(radius)-drawingPanel.
yToPix(0));
```

```
                g.setColor(Color.blue);
                for(int i = 0;i<Nd;i++) {
                        int xpix = drawingPanel.xToPix(x[i])-pxRadius;
                        int ypix = drawingPanel.yToPix(y[i])-pyRadius;
                        g.fillOval(xpix, ypix, 2*pxRadius, 2*pyRadius);
                }
                g.setColor(Color.red);
                for(int i = 0;i<Np;i++) {
                        pxRadiusP[i] = Math.abs(drawingPanel.xToPix(a[i])-
    drawingPanel.xToPix(0)); // Alan
                        pyRadiusP[i] = Math.abs(drawingPanel.yToPix(b[i])-
    drawingPanel.yToPix(0)); // Alan
                        int xpixP = drawingPanel.xToPix(xp[i])-pxRadiusP[i];
                        int ypixP = drawingPanel.yToPix(yp[i])-pyRadiusP[i];
                        Ellipse2D e = new Ellipse2D.Double(xpixP, ypixP, 2*pxRadiusP
    [i], 2*pyRadiusP[i]);
                        AffineTransform at = AffineTransform.getRotateInstance(theta
    [i], xpixP+pxRadiusP[i], ypixP+pyRadiusP[i]);
                        g2.fill(at.createTransformedShape(e));
                }
                g.setColor(Color.black);
                int xpix = drawingPanel.xToPix(0);
                int ypix = drawingPanel.yToPix(Ly);
                int lx = drawingPanel.xToPix(Lx)-drawingPanel.xToPix(0);
                int ly = drawingPanel.yToPix(0)-drawingPanel.yToPix(Ly);
                g.drawRect(xpix, ypix, lx, ly);
        }

}
```

A.5. 2D Monte-Carlo Simulation Main() Code

```
package org.opensourcephysics.sip.ch15;
import java.awt.*;
import java.awt.geom.Ellipse2D;
import java.awt.geom.AffineTransform;
import org.opensourcephysics.display.*;
import org.opensourcephysics.frames.*;
import org.opensourcephysics.numerics.*;
import java.util.stream.*;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.GUIUtils;
import java.io.*;
import java.io.FileWriter;
import java.util.List;
import java.util.ArrayList;
import java.util.Calendar;

/**
 * HardDisksApp does a Monte Carlo simulation for a 2D system of uncharged hard
   disks crowding an elliptical polymer
 *
 * @author Wyatt Davis & Alan Denton based on MD code ch08/hd/HardDisksApp.java by
   Jan Tobochnik, Wolfgang Christian, Harvey Gould
 * @version 1.5 revised 4/18
 *
 */



public class DPM5App extends AbstractSimulation {
        DPM5 hd = new DPM5();
        DisplayFrame display = new DisplayFrame("x", "y", "Hard Disk Particles");
        int i = 0; // Will track number of steps taken
        int j = 0; // Will track number of data points written
        int k = 0; // Will track number of independant runs
        Calendar cal = Calendar.getInstance();
```

```
    ArrayList<Double> L1dat = new ArrayList<Double>(10000);
    ArrayList<Double> L2dat = new ArrayList<Double>(10000);


    /**
    * Initializes the simulation
    */
    public void initialize() {
            hd.Nd = control.getInt("Nd");
            hd.Np = control.getInt("Np");
            hd.phi = control.getDouble("phi");
            hd.Lx = .5*Math.sqrt(hd.Nd*Math.PI/hd.phi);
            hd.Ly = .5*Math.sqrt(hd.Nd*Math.PI/hd.phi);
            hd.condition=false;
            hd.tolerance = control.getDouble("tolerance");
            String configuration = control.getString("initial configuration");
            hd.sizeRatio = control.getDouble("sizeRatio");
            hd.penetrationCost= control.getDouble("penetrationCost");
            hd.solventQuality = control.getString("solventQuality");
            hd.initialize(configuration);
            display.addDrawable(hd);
            display.setPreferredMinMax(0, hd.Lx, 0, hd.Ly);
            display.setSquareAspect(true);

    }

    //Used step method from  how data is taken over duration of simulation
    public void doStep() {
            if(k<6){
            if(k==0 && i==0){
                    File dir1 = new File("data/phiN="+hd.phi+",q="+hd.sizeRatio
+"_");
                    boolean successful1 = dir1.mkdir();
                    if (successful1){
                            System.out.println("directory was created
successfully");
                    }
                    else{
                            System.out.println("failed trying to create the
directory");
                    }
                    for(int i=1; i<6; i++){
                            File dir2 = new File("data/phiN="+hd.phi+",q="+hd.
sizeRatio+"_" +"/" + i);
                            boolean successful2 = dir2.mkdir();
                            if (successful2){
                                    System.out.println("directory was created
successfully");
                            }
                            else{
                                    System.out.println("failed trying to create
the directory");
                            }
                            try{
                                    FileWriter fw=new FileWriter("data/phiN="+hd
.phi+",q="+hd.sizeRatio+"_" +"/" + i +"/"+"phiN="+hd.phi+",q="+hd.sizeRatio+"_eX.
dat");
                            }catch(Exception e){System.out.println(e);}
                            System.out.println("Success...");
                            try{
                                    FileWriter fw=new FileWriter("data/phiN="+hd
.phi+",q="+hd.sizeRatio+"_" +"/" + i +"/"+"phiN="+hd.phi+",q="+hd.sizeRatio+"_eY.
dat");
                            }catch(Exception e){System.out.println(e);}
                            System.out.println("Success...");
                    }
            k++;
            }
        //check if enough datapoints have been taken
            if (j<10000){
            //allow steps for equilibriation
            if(i<10000){
                    hd.step();
                    i++;
                }
                //after steps, start data collection procedure
                else{
                    hd.step();
                    i++;
                    if (i%1000 == 0){
                        if(hd.l1[0]>=hd.l2[0]){
                            L1dat.add(hd.l1[0]);
                            L2dat.add(hd.l2[0]);
                        }
                        else{
                            L1dat.add(hd.l2[0]);
                            L2dat.add(hd.l1[0]);
                        }
                        j++;
                        //System.out.print(j);
                    }
                }
        }
  //WRITE DATA
                else{
                    String strFilePathX = "data/phiN="+hd.phi+",q="+hd.sizeRatio
+"_" +"/" + k +"/"+"phiN="+hd.phi+",q="+hd.sizeRatio+"_eX.dat";
                    String strFilePathY = "data/phiN="+hd.phi+",q="+hd.sizeRatio
+"_" +"/" + k +"/"+"phiN="+hd.phi+",q="+hd.sizeRatio+"_eY.dat";
                    System.out.println(strFilePathX);
                    try{
                        FileWriter fwX = new FileWriter(strFilePathX);
                        BufferedWriter bw = new BufferedWriter(fwX);
                        for (double dat: L1dat){
                            bw.write(dat+"");
                            bw.newLine();
                        }
                        bw.close();
                    }catch(Exception e){System.out.println(e);}
```

```
                        try{
                                FileWriter fwY = new FileWriter(strFilePathY);
                                BufferedWriter bw = new BufferedWriter(fwY);
                                for (double dat: L2dat){
                                        bw.write(dat+"");
                                        bw.newLine();
                                }
                                bw.close();
                        }catch(Exception e){System.out.println(e);}
                        k++;
                        L1dat.clear();
                        L2dat.clear();
                        i=0;
                        j=0;
                        initialize();
                        }
                }
        }

/**
 * Resets the hard disks model to its default state.
 */
        public void reset() {
                enableStepsPerDisplay(true);
                control.setValue("Nd", 36);
                control.setValue("Np",1);
                control.setValue("phi",.2);
                control.setValue("tolerance", 0.1);
                control.setValue("initial configuration", "regular");
                control.setValue("sizeRatio",1);
                control.setValue("penetrationCost",.5);
                control.setValue("solventQuality", "theta");
                initialize();
        }

/**
```

```
   * Resets the hard disks model and the data graphs.
   *
   * This method is invoked using a custom button.
   */
        public void resetData() {
                hd.resetAverages();
                GUIUtils.clearDrawingFrameData(false); // clears old data from the
    plot frames
        }

  /**
   * Starts the Java application.
   * @param args  command line parameters
   */
        public static void main(String[] args) { // set up animation control
    structure using this class
                SimulationControl control = SimulationControl.createApp(new DPM5App
    ());
                control.addButton("resetData", "Reset Data");

        }
}
```

A.6. Data Analysis Script for 2D RW

```
#!/usr/bin/env python
import sys
import argparse
from math import sqrt, pow, fabs, pi
from os import listdir, getcwd
from os.path import isdir, abspath, join

#------------------Functions----------------------

def is_number(n):
        try:
                float(n)
                return True
        except ValueError:
                return False

# Read data from simulation datafile, and parse it into a list
def read(path):
        with open(path, 'r') as f:
                ret = []
                for line in f:
                        token = line.rstrip()
                        if(is_number(token)):
                                ret.append(float(token))
                return ret

def histogram(data, bins, end):
        width = float(end) / float(bins)
        hist = dict()
        data = sorted(data)
        x = 0
        for i in range(0,bins+1):
                count = 0

                # Putting data points into bins
                while x < len(data) and data[x] <= width*i:
                        count += 1
                        x += 1
                hist[width*i] = count

        # Normalizing histogram
        area = 0
        for k,v in hist.iteritems():
                area += width*v
        normalize = 1 / float(area) # normalizing constant

        for k in hist.iterkeys():
                hist[k] *= normalize
        return hist

def min(x,y):
        if(x > y):
                return y
        else:
                return x

#redefined for 2D (RW)
def Vol(eX, eY):
        mlen = min(len(eX),len(eY))
        i = 0
        total = 0
        for i in range(mlen):
                total += sqrt(float(eX[i])*float(eY[i]))
        #return 4*pi*(1.9634**3)*total/3/float(mlen)
        return pi*total*12.0/float(mlen)

#redefined for 2D (RW))
def rgOverRgr(eX, eY):
        mlen = min(len(eX),len(eY))
        i = 0
        total = 0
        for i in range(mlen):
                total += float(eX[i]) + float(eY[i])
        #return sqrt(total/float(mlen))/.44108
        return sqrt(6*total/float(mlen))

#Continue
def asphericity(eX, eY):
        mlen = min(len(eX),len(eY))
        i = 0
        term1 = 0
        #term2 = 0
        #term3 = 0
        denom = 0
        for i in range(mlen):
                term1 += pow(eY[i] - eX[i], 2)
                #term2 += pow(eZ[i] - eX[i],
                #term3 += pow(eZ[i] - eY[i], 2)
                denom += pow(eX[i]+eY[i], 2)
        numerator = term1/float(mlen)
        denom = denom / float(mlen)
        return numerator / denom

def average(list):
        total = 0
        for a in list:
                if(is_number(a)):
                        total += a
        return total / float(len(list))

def stddev(list):
        return sqrt(average(map(lambda x : x**2, list)) - pow(average(list),2)) #
    Note : x**2 = x^2, x squared

def analyzeRun(runDirectory, grouping_dist):
        directory = runDirectory # Full file directory eg. phiN=0.8,q=0.5_12
    -21-6-2015 11:34
        prefix = directory.split("_")[0] # Prefix specifying phiN and q, eg. phiN
    =0.8,q=0.5
        phiN = (prefix.split(",")[0]).split("=")[1]
        datapoints = 5 # Number of data variables we're interested in
        basedir = abspath(join(getcwd(),directory))
        datadirs = [abspath(join(basedir,f)) for f in listdir(basedir) if isdir(
    abspath(join(basedir,f)))]
        results = [[] for x in range(datapoints)] # Create a list for each datapoint
        runs = len(datadirs)

        # Read all data from the runs, append it to results list.
        # Indices 0,1,2 will store eX,eY,eZ and 3,4 will store radius of gyration,
    asphericity.
        for d in datadirs:
                eXdata = read(join(d, '{}_eX.dat'.format(prefix)))
                eYdata = read(join(d, '{}_eY.dat'.format(prefix)))
                #eZdata = read(join(d, '{}_eZ.dat'.format(prefix)))
                #sizeData = read(join(d, '{}_R^3.dat'.format(prefix)))
                results[0].append(histogram(eXdata, 400, 0.7))
                results[1].append(histogram(eYdata, 200, 0.15))
                #results[2].append(histogram(eZdata, 200, 0.02))
                results[2].append(rgOverRgr(eXdata, eYdata))
                results[3].append(asphericity(eXdata, eYdata))
                results[4].append(Vol(eXdata, eYdata))

        i = 0
        for i in range(datapoints):
                if i == 0:
                        f = open(join(basedir,'eX.dat'), 'w')
                elif i == 1:
                        f = open(join(basedir,'eY.dat'), 'w')
                #elif i == 2:
                #   f = open(join(basedir,'eZ.dat'), 'w')
                elif i == 2: # Rg calculation
                        f = open(join(basedir,'rg.dat'), 'w')
                        rgr = results[2]
```

```
                f.write("{}\t{:.5f}\t{:.5f}".format(phiN, average(rgr),
stddev(rgr)))
            elif i == 3: # Asphericity calculation
                f = open(join(basedir,'aspher.dat'), 'w')
                aspher = results[3]
                f.write("{}\t{:.5f}\t{:.5f}".format(phiN, average(aspher),
stddev(aspher)))
            else:
                f = open(join(basedir, 'volume.dat'), 'w')
                #vol = map(lambda x : 4.1887*x, results[5])
                vol = results[4]
                f.write("{}\t{:.5f}\t{:.5f}".format(phiN, average(vol),
stddev(vol)))



            # Calculate average of histograms for eX, eY
            if i < 2:
                avgd_hist = dict() # to store averaged histogram after 5
runs
                hist = results[i][0] # get the keys (x-values) first

                # average the histograms
                for k in sorted(hist.keys()):   # for each bin
                    total = 0
                    sqr_total = 0
                    for j in range(runs):
                        total += results[i][j][k]
                        sqr_total += results[i][j][k]*results[i][j][
k]
                    avg = total / float(runs)
                    sqr_avg = sqr_total / float(runs)
                    avg_sqrd = avg * avg
                    std = sqrt(abs(sqr_avg-avg_sqrd))
                    avgd_hist[k] = [avg, std]

                # consolidating bins as necessary
```

```
                        keyList = sorted(avgd_hist.keys())
                        j = 0
                        # the grouping distance is adjusted based on the eX, eY, eZ
    curves
                        if i == 0:
                                gdist = grouping_dist
                        elif i == 1:
                                gdist = 4 * grouping_dist
                        #elif i == 2:
                        #       gdist = 10 * grouping_dist

                        while j < len(keyList):
                                init_y = avgd_hist[keyList[j]][0]
                                count = 1
                                total_y = init_y
                                total_std = avgd_hist[keyList[j]][1]
                                total_x = keyList[j]
                                while(j < len(keyList) and fabs(avgd_hist[keyList[j
    ]][0]-init_y) < gdist): # all bins that have their y-values separated with less
    than specified unit distance are grouped
                                        total_y += avgd_hist[keyList[j]][0]
                                        total_std += avgd_hist[keyList[j]][1]
                                        total_x += keyList[j]
                                        count += 1
                                        j +=1
                                consolidated_y = total_y / float(count)
                                consolidated_std = total_std / float(count)
                                consolidated_x = total_x / float(count)
                                f.write("{:.4f}\t{:.5f}\t{:.5f}\n".format(
    consolidated_x,consolidated_y, consolidated_std))
                                j += 1
                f.close()

def read_into_dict(file_path, insert_dict):
        f = open(file_path, 'r')
        line = f.readline()
        tokens = line.split("\t")
        key = tokens[0]
        value = tokens[1] + "\t" + tokens[2]
        insert_dict[float(key)] = value
        f.close()

def write_sorted(write_path, dict_to_write):
        f = open(write_path, "w")
        for key in sorted(dict_to_write):
                f.write("{:.5f}\t{}\n".format(key, dict_to_write[key]))
        f.close()



#----------------Main------------------

if __name__ == '__main__':
        parser = argparse.ArgumentParser()
        parser.add_argument("-r", "--run", help="Analyze only specific run directory
    .")
        parser.add_argument("group_dist", type=float,help="Minimum histogram y-
    grouping distance.")
        args = parser.parse_args()
        if args.run:
                analyzeRun(args.run, args.group_dist)
        else:
                run_dirs = [f for f in listdir(getcwd()) if isdir(abspath(join(
    getcwd(),f)))]
                aspher = {}
                rg = {}
                vol = {}
                for run_dir in run_dirs:
                        # Perform analysis
                        analyzeRun(run_dir, args.group_dist)
                        # Combine data
                        read_into_dict(join(run_dir, "rg.dat"), rg)
                        read_into_dict(join(run_dir, "aspher.dat"), aspher)
```

```
            read_into_dict(join(run_dir, "volume.dat"), vol)


    # Write combined data
    write_sorted("rg.dat", rg)
    write_sorted("aspher.dat", aspher)
    write_sorted("vol.dat", vol)
```

A.7. Data Analysis Script for 2D SAW

```
#!/usr/bin/env python
import sys
import argparse
from math import sqrt, pow, fabs, pi
from os import listdir, getcwd
from os.path import isdir, abspath, join

#-----------------Functions---------------------

def is_number(n):
        try:
                float(n)
                return True
        except ValueError:
                return False

# Read data from simulation datafile, and parse it into a list
def read(path):
        with open(path, 'r') as f:
                ret = []
                for line in f:
                        token = line.rstrip()
                        if(is_number(token)):
                                ret.append(float(token))
                return ret

def histogram(data, bins, end):
        width = float(end) / float(bins)
        hist = dict()
        data = sorted(data)
        x = 0
        for i in range(0,bins+1):
                count = 0

                # Putting data points into bins
                while x < len(data) and data[x] <= width*i:
                        count += 1
                        x += 1
                hist[width*i] = count

        # Normalizing histogram
        area = 0
        for k,v in hist.iteritems():
                area += width*v
        normalize = 1 / float(area) # normalizing constant

        for k in hist.iterkeys():
                hist[k] *= normalize
        return hist

def min(x,y):
        if(x > y):
                return y
        else:
                return x

#redefined for 2D (SAW)
def Vol(eX, eY):
        mlen = min(len(eX),len(eY))
        i = 0
        total = 0
        for i in range(mlen):
                total += sqrt(float(eX[i])*float(eY[i]))
        #return 4*pi*(1.9634**3)*total/3/float(mlen)
        #return pi*total*12.0/float(mlen)
        return pi*total*18.492797256/float(mlen)
#redefined for 2D (RW))
def rgOverRgr(eX, eY):
        mlen = min(len(eX),len(eY))
        i = 0
        total = 0
        for i in range(mlen):
                total += float(eX[i]) + float(eY[i])
        #return sqrt(total/float(mlen))/.44108
        return (1/.328862)*sqrt(total/float(mlen))

#Continue
def asphericity(eX, eY):
        mlen = min(len(eX),len(eY))
        i = 0
        term1 = 0
        #term2 = 0
        #term3 = 0
        denom = 0
        for i in range(mlen):
                term1 += pow(eY[i] - eX[i], 2)
                #term2 += pow(eZ[i] - eX[i],
                #term3 += pow(eZ[i] - eY[i], 2)
                denom += pow(eX[i]+eY[i], 2)
        numerator = term1/float(mlen)
        denom = denom / float(mlen)
        return numerator / denom

def average(list):
        total = 0
        for a in list:
                if(is_number(a)):
                        total += a
        return total / float(len(list))

def stddev(list):
        return sqrt(average(map(lambda x : x**2, list)) - pow(average(list),2)) #
    Note : x**2 = x^2, x squared

def analyzeRun(runDirectory, grouping_dist):
        directory = runDirectory # Full file directory eg. phiN=0.8,q=0.5_12
    -21-6-2015 11:34
        prefix = directory.split("_")[0] # Prefix specifying phiN and q, eg. phiN
    =0.8,q=0.5
        phiN = (prefix.split(",")[0]).split("=")[1]
        datapoints = 5 # Number of data variables we're interested in
        basedir = abspath(join(getcwd(),directory))
        datadirs = [abspath(join(basedir,f)) for f in listdir(basedir) if isdir(
    abspath(join(basedir,f)))]
        results = [[] for x in range(datapoints)] # Create a list for each datapoint
        runs = len(datadirs)

        # Read all data from the runs, append it to results list.
        # Indices 0,1,2 will store eX,eY,eZ and 3,4 will store radius of gyration,
    asphericity.
        for d in datadirs:
                eXdata = read(join(d, '{}_eX.dat'.format(prefix)))
                eYdata = read(join(d, '{}_eY.dat'.format(prefix)))
                #eZdata = read(join(d, '{}_eZ.dat'.format(prefix)))
                #sizeData = read(join(d, '{}_R^3.dat'.format(prefix)))
                results[0].append(histogram(eXdata, 400, 0.7))
                results[1].append(histogram(eYdata, 200, 0.15))
                #results[2].append(histogram(eZdata, 200, 0.02))
                results[2].append(rgOverRgr(eXdata, eYdata))
                results[3].append(asphericity(eXdata, eYdata))
                results[4].append(Vol(eXdata, eYdata))

        i = 0
        for i in range(datapoints):
                if i == 0:
                        f = open(join(basedir,'eX.dat'), 'w')
                elif i == 1:
                        f = open(join(basedir,'eY.dat'), 'w')
                #elif i == 2:
                #   f = open(join(basedir,'eZ.dat'), 'w')
                elif i == 2: # Rg calculation
                        f = open(join(basedir,'rg.dat'), 'w')
                        rgr = results[2]
```

```
                f.write("{}\t{:.5f}\t{:.5f}".format(phiN, average(rgr),
stddev(rgr)))
            elif i == 3: # Asphericity calculation
                f = open(join(basedir,'aspher.dat'), 'w')
                aspher = results[3]
                f.write("{}\t{:.5f}\t{:.5f}".format(phiN, average(aspher),
stddev(aspher)))
            else:
                f = open(join(basedir, 'volume.dat'), 'w')
                #vol = map(lambda x : 4.1887*x, results[5])
                vol = results[4]
                f.write("{}\t{:.5f}\t{:.5f}".format(phiN, average(vol),
stddev(vol)))



            # Calculate average of histograms for eX, eY
            if i < 2:
                avgd_hist = dict() # to store averaged histogram after 5
runs
                hist = results[i][0] # get the keys (x-values) first

                # average the histograms
                for k in sorted(hist.keys()):   # for each bin
                    total = 0
                    sqr_total = 0
                    for j in range(runs):
                        total += results[i][j][k]
                        sqr_total += results[i][j][k]*results[i][j][
k]
                    avg = total / float(runs)
                    sqr_avg = sqr_total / float(runs)
                    avg_sqrd = avg * avg
                    std = sqrt(abs(sqr_avg-avg_sqrd))
                    avgd_hist[k] = [avg, std]

                # consolidating bins as necessary
            keyList = sorted(avgd_hist.keys())
            j = 0
            # the grouping distance is adjusted based on the eX, eY, eZ
  curves
            if i == 0:
                gdist = grouping_dist
            elif i == 1:
                gdist = 4 * grouping_dist
            #elif i == 2:
            #       gdist = 10 * grouping_dist

            while j < len(keyList):
                init_y = avgd_hist[keyList[j]][0]
                count = 1
                total_y = init_y
                total_std = avgd_hist[keyList[j]][1]
                total_x = keyList[j]
                while(j < len(keyList) and fabs(avgd_hist[keyList[j
  ]][0]-init_y) < gdist): # all bins that have their y-values separated with less
  than specified unit distance are grouped
                    total_y += avgd_hist[keyList[j]][0]
                    total_std += avgd_hist[keyList[j]][1]
                    total_x += keyList[j]
                    count += 1
                    j +=1
                consolidated_y = total_y / float(count)
                consolidated_std = total_std / float(count)
                consolidated_x = total_x / float(count)
                f.write("{:.4f}\t{:.5f}\t{:.5f}\n".format(
  consolidated_x,consolidated_y, consolidated_std))
                j += 1
        f.close()

def read_into_dict(file_path, insert_dict):
    f = open(file_path, 'r')
    line = f.readline()
```

```
        tokens = line.split("\t")
        key = tokens[0]
        value = tokens[1] + "\t" + tokens[2]
        insert_dict[float(key)] = value
        f.close()

def write_sorted(write_path, dict_to_write):
        f = open(write_path, "w")
        for key in sorted(dict_to_write):
                f.write("{:.5f}\t{}\n".format(key, dict_to_write[key]))
        f.close()



#----------------Main-----------------

if __name__ == '__main__':
        parser = argparse.ArgumentParser()
        parser.add_argument("-r", "--run", help="Analyze only specific run directory
    .")
        parser.add_argument("group_dist", type=float,help="Minimum histogram y-
    grouping distance.")
        args = parser.parse_args()
        if args.run:
                analyzeRun(args.run, args.group_dist)
        else:
                run_dirs = [f for f in listdir(getcwd()) if isdir(abspath(join(
    getcwd(),f)))]
                aspher = {}
                rg = {}
                vol = {}
                for run_dir in run_dirs:
                        # Perform analysis
                        analyzeRun(run_dir, args.group_dist)
                        # Combine data
                        read_into_dict(join(run_dir, "rg.dat"), rg)
                        read_into_dict(join(run_dir, "aspher.dat"), aspher)
```

```
            read_into_dict(join(run_dir, "volume.dat"), vol)


    # Write combined data
    write_sorted("rg.dat", rg)
    write_sorted("aspher.dat", aspher)
    write_sorted("vol.dat", vol)
```