Modelling Endocytic Actin Bundling

Beth McMillan
MSc Mechanistic Biology Research Project

Supervisor: Rhoda Hawkins

Summary

Endocytosis is a process that occurs in all eukaryotic cells, in which the membrane invaginates to take in cargo from outside the cell. The structural protein actin is essential for endocytosis, as is fimbrin, a protein which bundles actin filaments together into larger assemblies.

This report describes an agent-based Monte Carlo simulation of tethered actin filaments polymerising against a membrane barrier in order to elongate the membrane invagination. The model is one-dimensional, calculating the length of the membrane invagination over time.

In the typical course of a run, the filaments grow from a tether at the tip of the invagination until they reach the outer membrane of the cell, which prevents them from elongating. A brownian ratchet mechanism is proposed for allowing filaments to polymerise against the membrane, elongating the invagination. The filaments grow, shrink, and bend. Bundling proteins attach to the filaments, affecting the amount of bending.

The program is written in Python, with the addition of the “PyGame” and “PyX” modules, as well as “math”, “random” and “sys”. The code consists of four core modules, with three additional programs for visualisation, and one modification of the main program to allow for running of several simulations with different parameters. Along with data files and graphs, the simulation outputs snapshots and animations of the process (Figure 1).

Table of contents

List of abbreviations

Abbreviation	**Description**
F-actin	Filamentous actin
G-actin	Globular actin
kDa	KiloDaltons
ATP	Adenosine TriPhosphate
ADP	Adenosine DiPhosphate
Pi	Inorganic Phosphate
Arp2/3	Actin-Related Proteins 2 and 3
Las17p	Local Anaestheticum Sensitive Protein 17
Ent1p	Epsin N-Terminal Homology Protein 1
Ent3p	Epsin N-Terminal Homology Protein 3
Sla1p	Synthetic Lethal with ABP1 (Actin Binding Protein 1) Protein 1
Sla2p	Synthetic Lethal with ABP1 (Actin Binding Protein 1) Protein 2
Pan1p	Poly(A)-binding protein-dependent poly(A) riboNuclease 1
PIP_2	PhosphatidylInositol-4,5-bisPhosphate
Myo3p	Myosin 3
Myo5p	Myosin 5
Bzz1p	Las17p (Bee1p) interacting protein
Cap1p	Capping Protein 1
Cap2p	Capping Protein 2
Sac6p	Suppressor of ACtin 6
Rvs161p	Reduced Viability on Starvation Protein 161

Rvs167p	Reduced Viability on Starvation Protein 167
BAR	Bin/Amphiphysin/Rvs
PIP-5 kinase	PhosphotidylInositol-4-Phosphate 5-kinase

1. Introduction

Actin is an abundant structural protein, which is involved in many processes inside eukaryotic cells, including cell division, endocytosis, and cellular motility. Actin monomers are known as globular, or G-actin. They assemble in a helical formation to create long assemblies, called filamentous, or F-actin. These filaments are continually growing and disassembling. F-actin is used to support structures in the cell, to provide force for movement, and as tracks for transport. Actin is a key player in the cellular process of endocytosis (Pollard & Cooper, 2009).

In endocytosis, the membrane of a cell invaginates to take in cargo from outside (see Figure 2). First, the membrane is deformed to create a pit (1), and then a tubular structure (2). Once the tube reaches around 40nm in length, a spherical membrane-bound vesicle buds off from the end (3) and delivers the cargo to be processed in the endo-lysosomal membrane system inside the cell (4). Proteins are recruited sequentially during the process, with defined times for the arrival and departure of each (Kaksonen, 2008). In yeast, endocytosis is used to pass on signals, to recycle the membrane, and to take up nutrients (Liu et al, 2006).

During endocytosis, actin is found near the cell surface in dynamic assemblies known as “actin patches”. When yeast cells are mutated to lack actin, or when actin is sequestered by a drug (such as latrunculin A), endocytosis does not take place (Smythe & Ayscough, 2006). While it is clear that actin is required for endocytosis in yeast, it is not known exactly what role it plays. In other actin-mediated processes, including cell motility, F-actin polymerising against a barrier can work to drive motion (Pollard & Borisy, 2003). Actin filaments are bundled into larger assemblies by fimbrin. Bundles are less likely to bend than single filaments, and therefore could be more effective in driving cellular events.

A Monte Carlo model is a computer simulation in which random numbers are used to dictate events. An agent based model is a simulation that casts each thing involved in it as an individual agent, with internal rules and variables, which interact with each other by sending messages.

I have created an agent-based Monte Carlo model of actin filaments growing in one dimension, and used it to investigate the mechanism of endocytosis – specifically, to answer the question: what effect do the numbers of membrane proteins have on the chances of successful endocytosis?

In this introduction, I will first discuss the process of endocytosis, folowed by the structure and dynamics of actin filaments, and finally the mathematical technique of Boltzmann weighting.

1.1. The process of endocytosis

1.1.1. Membrane deformation and coat complex formation

The first step in endocytosis is the initial deformation of the membrane by the endocytic coat complex. This complex includes the yeast Epsin homologues Ent1p and Ent3p, which interact with membrane lipids to drive initial inwards curvature of the membrane. Ent1p and Ent3p also attach to cargo: they bind membrane receptors that have been tagged with ubiquitin. Ubiquitin-tagging is used by cells to mark proteins for recycling or degradation (Shih et al 2002).

The endocytic coat complex also contains clathrin, which maintains membrane curvature by forming a polyhedral basket-like structure (Smythe & Ayscough 2006). Proteins which affect actin filament formation make up part of the endocytic coat complex: Sla1p, which prevents actin nucleating by down-regulating Las17p, and binds to cargo proteins; Sla2p, which interacts with the membrane and with actin directly; and Pan1p, which activates Arp2/3. The coat complex acts as an anchor and regulator of actin filaments (Liu et al, 2009)

The uncommon membrane phospholipid PIP_2 (phosphatidylinositol-4,5-bisphosphate) is also thought to be a key component in the endocytic machinery. Endocytic coat proteins, including Sla2p, have PIP_2 binding domains, and the removal of these binding domains impairs endocytosis. This indicates that PIP_2 helps to recruit proteins to sites of endocytosis (Sun et al, 2007). PIP_2 is produced by phosphorylation of membrane lipids by lipid kinases, and it is degraded by lipid phosphatises and lipases. The amount of PIP_2 in the endocytic site peaks during vesicle internalisation and then decreases towards the end of the process (Liu et al, 2009).

When the components of the endocytic coat complex are fluorescently labelled, their behaviour during endocytosis can be observed. The coat complex constituents accumulate as static fluorescent spots on the plasma membrane, which then move around 200nm into the cell and disassemble. This suggests that these proteins localise to the tip of the membrane invagination (Kaksonen et al, 2003).

1.1.2. Actin nucleation

Las17p is the first protein to arrive at the endocytic site, and it remains at the membrane throughout endocytosis, initially kept inactivated by the coat complex protein Sla1p. Mutant yeast lacking Las17p do not form actin patches at the cell cortex (Li, 1997) and are incapable of endocytosis (Navqi et al, 1998). Las17p recruits verprolin, which in turn recruits two type I myosins: Myo3p and Myo5p. These bind to Arp2/3 and stimulate its nucleation activity (Kaksonen, 2010).

Las17p also recruits Bzz1p, which prevents Las17p inhibition by Sla1p, allowing Las17p to activate Arp2/3, initiating actin nucleation. Once initiated, filaments can then begin to form, creating the actin patch (Smythe & Ayscough, 2006).

1.1.3. Actin patch dynamics

Filaments form with the barbed end facing towards the membrane. The cap1p and cap2p heterodimer associates to cap actin filaments at the barbed end, preventing further monomer addition. This increases the strength of the actin network by keeping the filaments short, and ensures that new monomers are added only to recently-nucleated filaments (Kim et al, 2006). The bundling protein fimbrin attaches filaments together to make stronger assemblies.

1.1.4. Vesicle scission

Yeast amphiphysin homologues Rvs161p and Rvs167p are recruited to the site of endocytosis by actin, Las17p and constituents of the endocytic coat complex, soon after actin arrives. Rvs161p and Rvs167p contain BAR (Bin/Amphiphysin/Rvs) domains, which are known to bind to and tubulate membranes in vitro (Smythe & Ayscough 2006). BAR domain-containing proteins are also capable of sensing the curvature of membranes, so it is unclear whether the membrane tubulation is caused by these proteins or if the curvature induced by tubulation merely attracts them (Liu et al, 2009).

From the end of this tubule of membrane, the vesicle is formed. The mechanisms which provide the force to pinch off the vesicle from the end of the tubule are not fully characterised. A model proposed by Liu et al suggests that phase separation of lipids within the membrane, caused by interactions with tubule coat proteins, causes the interface between the vesicle and the tubule to shrink until it can be closed by thermal fluctuations in the membrane (Liu et al, 2006). However, this mechanism still requires additional force, presumably from actin filaments, to create vesicles the size seen *in vivo*.

1.1.5. Movement

The vesicle then moves through the cell to the endo-lysosomal membrane system. There are two competing theories for the role of actin in this process: that the vesicle is pulled along actin cables by motor proteins (Kim et al, 2006), or that it is propelled by nucleation of actin at the vesicle surface in a mechanism similar to the "comet tails" employed by the bacteria *Listeria monocytogenes* when moving inside host cells (Lambrechts et al, 2008).

1.2. Actin filament formation *in vitro*

Monomeric G-actin is a globular protein that weighs 42kDa. It consists of a single polypeptide chain with 375 amino acids. X-ray crystallography data shows that the protein has two major domains, which contain a mixture of alpha helices and beta sheets (see Figure 3). These can be divided further into four subdomains: subdomains 1 and 2 form the smaller domain, and subdomains 3 and 4 the larger. Subdomains 1 and 3 are structurally similar, possibly due to a gene duplication event (Dominguez & Holmes, 2011).

There are two clefts between the major domains: the upper cleft holds the nucleotide adenosine triphosphate (ATP) or the hydrolysed form adenosine diphosphate (ADP) with or without inorganic phosphate (Pi), as well as Mg^{2+} or another divalent cation bound to the nucleotide. The lower, hydrophobic cleft is involved in interactions between adjacent monomers in actin filaments, as well as binding to other proteins (Dominguez & Holmes, 2011).

When G-actin polymerises to form F-actin, the smaller domain is found on the outside of the filament, and the larger on the inside. Each monomer has one "pointed" end and one

“barbed” end. Monomers assemble end-to-end in F-actin, leaving each entire filament structurally polarised, with one barbed and one pointed end. Filamentous actin takes the form of a right-handed long-pitch double helix, with approximately thirteen monomers every six turns.

1.2.1. Treadmilling

F-actin is not a static molecule, and is in a continual state of dynamic growth and disassembly. G-actin is added to and removed from the filament at both ends at different rates – the barbed end has a lower “critical concentration”, meaning it grows more quickly, and monomers dissociate more quickly from the pointed end. For this reason, the barbed end is also known as the “plus” end, and the pointed end the “minus”. Canonically, actin filaments move through the cytoplasm by gaining monomers at the barbed end whilst losing monomers at the pointed end in a process known as “treadmilling”, in which the filament as a whole moves, but each individual monomer inside the filament remains stationary (see Figure 4).

G-actin is not an effective ATPase, and is usually found bound to ATP rather than ADP. Polymerisation into filaments stimulates the ATPase activity of actin monomers and allows them to hydrolyse ATP at a rate of 0.3 molecules per second (Fujiwara et al, 2007). Conformational changes on ATP hydrolysis and phosphate release are communicated from the upper to the lower cleft, causing larger conformational changes in the molecule. ADP-bound actin monomers dissociate more quickly from the filament than ATP-bound monomers. In general, monomers join the filament at the barbed end bound to ATP, hydrolysis of ATP and dissociation of Pi occurs inside the filament, and then ADP-bound monomers dissociate at the pointed end (Dominguez & Holmes, 2011).

1.2.2. Young's modulus

The Young's modulus of a material is a measure of its stiffness. It is defined as the tensile stress (i.e. stress due to stretching) divided by the tensile strain.

The tensile stress (σ) is defined as the force exerted on an object divided by the object's cross-sectional area.

$$\sigma = \frac{F}{A}$$

Tensile strain is a dimensionless property that describes the amount the material is stretched or compressed. For a beam that begins with length L_0 and is stretched or compressed to length $L_0 + \Delta L$, the tensile strain (ε) is:

$$\varepsilon = \frac{\Delta L}{L_0}$$

The Young's modulus (Y) can therefore be described by the equation:

$$Y = \frac{F L_0}{A \Delta L}$$

An actin filament has a Young's modulus of 1.8×10^9 N nm^{-2} (Kojima et al, 1994).

1.3. Actin filament formation *in vivo* in yeast

Although studying the dynamics of actin filament formation and disassembly in isolation has brought insights into the action of actin *in vivo*, these insights do not comprise the entire picture. In cells, the dynamic regulation of actin filament formation and recycling is mediated by a large number of actin binding proteins which nucleate, cap and sever F-actin, as well as re-activating G-actin for use in new filaments. Actin binding proteins are also themselves controlled by other proteins such as kinases, creating a complex multi-layer regulatory system which alters the kinetics from those of the reactions seen *in vitro*.

1.3.1. Nucleation

Since G-actin dimers are very stable, the formation of G-actin trimers is a limiting step in the spontaneous formation of actin filaments. In vivo, new filaments are created by nucleation factors, such as Arp2/3, which is a multi-subunit complex that contains, amongst other proteins, two subunits that are structurally related to actin.

When activated, the complex binds to the pointed end of an actin monomer and to the side of an existing actin filament, and stimulates polymerisation towards the barbed end at a 78˚ angle to the mother filament, creating branched structures (Pollard & Berro, 2009). The actin nucleation activity of Arp2/3 is regulated by several other proteins, most importantly the activator Las17p (also known as Bee1), which plays a key role in endocytosis (Winter et al, 1999).

1.3.2. Severing

Filaments are severed by cofilin, which binds to ADP-actin towards the pointed end of the filament and increases the depolymerisation rate 22-fold. This maintains a pool of G-actin monomers that can be re-used in new filaments (Theriot, 1997). Mathematical modelling has suggested that cofilin may act by creating short filaments that diffuse away from the site of actin polymerisation (Berro et al, 2010)

1.3.3. Capping

Capping proteins prevent further elongation of actin filaments by binding to the barbed ends. Capping of filaments is essential in many processes which require force from actin filaments (Kim et al, 2006), as it allows filament elongation to be concentrated to only certain areas of the branched network. The pool of active G-actin monomers is maintained by profilin, which binds to ADP-actin and catalyses the exchange of ADP for ATP, activating the monomer so it can be used in another filament (Pollard & Borisy, 2003).

1.3.4. Bundling

The bundling protein fimbrin (also called Sac6p, which stands for Suppressor of ACtin protein 6) interacts with filaments in order to bundle them together. Fimbrin has two actin-binding domains, which each bind to a filament. This makes strong structures, which are less likely to bend than single filaments (See Figure 5).

An experiment by Skau et al (2011), in which cells with a mutant version of fimbrin with only one actin binding domain were examined, showed the same phenotype as cells with no fimbrin at all. This indicates that it is specificially the bundling activity of fimbrin that is necessary for the cortical actin patch to assemble (Skau et al, 2011).

1.3.5. Brownian ratchet mechanism

When a filament grows until it is touching the outer membrane of the cell, it can continue to grow and either push the membrane outwards, or push itself backwards. This mechanism is known as a Brownian ratchet (see Figure 6). The cell membrane and the filament are both continually being buffeted by water molecules that are moving due to their thermal energy. This means that occasionally there is enough space between the filament's end and the membrane for a monomer to diffuse in and attach to the end of

the filament. In this way, the filament can continue growing even when it hits the membrane.

1.4 Boltzmann weighting

In this simulation, the probabilities of certain events are calculated using Boltzmann weighting. These events include actin filaments bending and the membrane being pushed forwards by a filament. For each of these events, the state in question (*i,* for example a bent filament) will have a particular energy (E_i). The Boltzmann factor is the relative probability of the particle being in state *i*, compared to a state with zero energy. The Boltzmann factor is given by the equation:

$$e^{\frac{-E_i}{k_B T}}$$

Where k_B is Boltzmann's constant and T is the temperature.

In order to calculate the absolute probability of the particle being in state *i*, the Boltzmann factor for state *i* needs to be normalised by dividing it by the Boltzmann factors for all of the possible states.

$$P(i) = N e^{\frac{-E}{k_B T}}$$

$$\frac{1}{N} = \int_0^\infty e^{\frac{-E}{k_B T}} dE$$

In the next section, I will discuss how this technique has been used to calculate the probabilities used in the model.

2. Materials and Methods

2.1. Introduction

This simulation is concerned with the elongation step of endocytosis. In this model, a group of filaments grow from the tip of a membrane invagination towards the outer membrane of the cell. When a filament reaches the membrane, it can either stop polymerising, bend, or push the membrane away, deepening the invagination. Bundling proteins attach the filaments together in pairs, creating larger bundles of filaments, which are less likely to bend.

The simulation tracks the length of the filaments and the length of the growing membrane invagination, making it a one-dimensional simulation. Each filament and bundling protein is modelled as an individual agent. The cell membrane also acts as a single agent. This is a Monte Carlo simulation, meaning that the steps in the model depend on the creation of (pseudo) random numbers by the computer.

First, I will describe the actions of each agent using the flow charts in Figures 7-9. Then I will discuss the equations used to calculate the parameters used in the simulation.

The model is written in Python, with the addition of the “math”, “sys” and “random” modules for extra functionality. The graphics were created using the PyX and PyGame modules. The term “random number” used from now on will refer to the pseudo random numbers generated by the “random” module.

The code is attached in Appendix 1. The main program “main.py” is accompanied by three modules, “functions.py”, “classes.py” and “parameters.py”, which contain the agent actions, definitions and variables, respectively. “Repeatedmain.py” is a variation on the main program that allows it to be called repeatedly. I have written four additional graphics programs: “graphs.py” creates graphs from the simulation output, “images.py” creates postscript images of the filaments and membrane, “animation.sh” creates animations from these images, and “newimages.py” creates a PyGame animation of the simulation.

2.2. Description of agents

The membrane agent represents the tip of the invagination, which is being pushed inside the cell by the actin filaments. Figure 7 shows a flow chart of the decisions made by the membrane agent at each time step. The membrane agent is the simplest of the three types of agent, with only one function: when the invagination has been lengthened, the membrane agent recieves information from the filament agents and changes its position.

The bundling protein agents (Figure 8) use a random number generator to decide whether or not to attach to two filaments. If the random number is over the threshold probability for attaching, it will use two further random numbers to pick two filaments and a position. In this simulation, the probability of a bundling protein attaching is the same for each filament and each position along the filaments.

The filament agents (Figure 9) first decide whether to polymerise or depolymerise, based on a random number. In order to polymerise, the filament must first check that it is not touching the outer membrane. If the filament is touching the membrane, a second random number is generated to decide whether or not to polymerise, and push the tip of the invagination further inside the cell, sending a message to the membrane agent to change position. If the filament does not move the membrane, a further random number is generated to decide whether or not the filament should bend. The bending probability depends on the length of the filament, the work done by the membrane pushing back agains the filament, and whether or not the filament is attached to bundling proteins.

2.3. Actin filament dynamics

I will now discuss the parameters used in this simulation, and the way in which they were derived.

2.3.1. Polymerisation and depolymerisation

The actin filaments are modelled as being tethered at their pointed ends, with all polymerisation and depolymerisation occurring at the barbed ends only. The rate of addition of new monomers to barbed end of each actin filament is based on the rate constant from Fujiwara et al, 2007.

$$k_{on} = 11.6\,\mu M^{-1} s^{-1}$$

In Sirotkin et al, 2010, the concentration of actin monomers around the site of endocytosis is given as:

$$[G-actin]=22\mu M$$

In this simulation, it is assumed that the concentration of G-actin remains constant. The probability of a given actin filament gaining a monomer at a given time step of length t seconds can therefore be described by the equation:

$$P_{on}=t\times k_{on}\times[G-actin]$$

Since the time step is 1ms (0.001 seconds), P_{on} can be calculated as:

$$P_{on}=0.001\times 11.6\times 22$$

$$P_{on}=0.2552$$

The probability of each filament losing a monomer from its barbed end is calculated similarly, using the value from Fujiwara et al, 2007.

$$k_{off}=0.25s^{-1}$$

$$P_{off}=0.001\times 0.25$$

$$P_{off}=0.00025$$

In the simulation, for each actin filament, at each time step, a pseudo-random number between 0 and 1 is generated. If the number is smaller than P_{on}, the length of the filament is increased by the length of one monomer (2.7nm). If the number is between P_{on} and $P_{on}+P_{off}$, the length of the filament is decreased by the length of one monomer. If the number is above $P_{on}+P_{off}$, the length of the filament stays the same.

2.3.2. Nucleation

In Beltzner and Pollard, 2008, the rate of nucleation of new actin filaments is given as:

$$k_{nuc} = 8 \times 10^{-9} \mu M^{-3} s^{-1}$$

Since three actin monomers are required to begin a new filament, the probability of a new filament being created in a given time step of length t seconds can be found by:

$$P_{nuc} = k_{nuc} \times [G-actin]^3 \times t$$

Using the value of G-actin concentration from Sirotkin et al, 2010, the probability is:

$$P_{nuc} = 8 \times 10^{-9} \times 22^3 \times 0.001$$

$$P_{nuc} = 7.51 \times 10^{-16}$$

At each time step, a random number between 1 and 0 is generated. If this number is below P_{nuc}, a new filament object is created with the length of three monomers (8.1nm).

2.3.3. Bending

In this simulation, when a filament is growing towards the membrane, there is a certain probability that the filament will bend, either by itself or as part of a bundle. This probability depends on several factors, including the length of the filament and the radius of the bundle.

The energy cost of bending (E) is calculated using the equation:

$$E = Y l i C^2$$

where Y is the Young's modulus of an actin filament, l is the length of the filament, i is the moment of inertia and C is the curvature.

The Young's modulus (Y) is a measure of the stiffness of the filament. In Kojima et al, 1994, the Young's modulus of an actin filament is given as:

$$Y = 1.8 \times 10^9 \, N \, m^{-2}$$

The moment of inertia (*i*) is calculated using the equation:

$$i = \frac{\pi r^4}{4}$$

Where *r* is the radius of the filament, or bundle of filaments. A single actin filament has a radius of 3.5nm. For larger bundles, the radius is 3.5nm multiplied by the number of filaments in the bundle.

When a filament bends, the curve that it forms can be imagined as being part of a larger circle. The radius of this imaginary circle is called the *radius of curvature* (R), and the reciprocal of this radius (i.e. $\frac{1}{R}$) is called the curvature (C).

The probability of bending to a particular curvature (P_{bend}) is found using the equation:

$$P_{bend} = N e^{\frac{Fx - E}{k_B T}}$$

Where N is a normalisation factor, F is the force felt by the filament due to the membrane, x is the distance the membrane has been pushed away from its starting point, k_B is Boltzmann's constant and T is the temperature.

The normalisation factor, N, is found by integrating the exponential function over the entire range of possible curvatures. This means that the probability will always be between 0 and 1.

$$\frac{1}{N} = \int_0^\infty e^{\frac{Fx - Y l i C^2}{k_B T}} dC$$

$$\frac{1}{N} = \frac{e^{\frac{Fx}{k_B T}} \sqrt{k_B T \pi}}{2\sqrt{Y l i}}$$

This bending is incorporated into the simulation by simply reducing the length of the filament. The horizontal component of the change will be ignored in this one-dimensional simulation.

For a filament that starts at length $L_{straight}$, the new length after bending to curvature C will be:

$$L_{bent} = \frac{1}{C} \sin(L_{straight} C)$$

At each time step, if the membrane has moved away, the filament will straighten, increasing in length, until it is at $L_{straight}$ or hits the membrane again.

2.4. Bundling protein dynamics

Bundling proteins attach two filaments together. In this simulation, the position of the bundling protein along the filaments is entirely random, as is the choice of which two filaments it attaches to. Each filament and each position has an equal chance of being selected.

2.5. Membrane dynamics

Since this is a one-dimensional simulation, the membrane is modelled as a single point. When a filament grows long enough to touch the membrane, the probability that the membrane will be pushed by the filament (P_{push}) is given by the equation:

$$P_{push} = P_{on} \times N e^{\frac{-\phi x^2}{k_B T}}$$

where P_{on} is the on-rate for polymerisation, N is a scaling factor, φ is the membrane tension, x is the distance the membrane has been pushed already, k_B is Boltzmann's constant and T is the temperature.

Similarly to the calculation for the filament bending probability, the scaling factor (N) is calculated by finding the integral of the exponential function between 0 and infinity, with respect to the distance the membrane has been pushed.

$$\frac{1}{N} = \int_0^{\infty} e^{\frac{-\phi x^2}{k_B T}} dx$$

$$\frac{1}{N} = \frac{1}{2}\sqrt{\frac{k_B T \pi}{\phi}}$$

This normalisation ensures that the probability will be between 0 and 1.

In the next section, I will discuss the results from repeated running of the simulation, along with the specific parameters that were used to obtain these results.

3. Results

The simulation was run repeatedly with between 0 and 59 bundling proteins, and the time taken for the invagination to reach 40nm was measured. Graphs of the results are shown in Figures 10-13.

Figures 10 and 11 show results for a curvature of 0.001nm^{-1}, while Figures 12 and 13 show results for a larger curvature of 0.01nm^{-1}. Each point in Figures 10 and 13 shows the mean of 20 runs of the simulation, while each point in Figures 11 and 13 shows a single run of the simulation.

The other parameters used in the model are in the table below

Parameter	Value	Units	Reference
Membrane tension	1 x 10^{-13}	N nm^{-1}	Liu et al, 2006
Number of filaments	10	N/a	N/a
Young's modulus	1.8 x 10^{-9}	N nm^{-2}	Kojima et al, 1994
Temperature	298	K	N/a
G-actin concentration	22	μM	Sirotkin et al, 2010
Length of a single monomer	2.7	nm	Dominguez & Holmes, 2011
Probability of adding a monomer	0.25520	ms^{-1}	Fujiwara et al 2007
Probability of losing a monomer	0.00025	ms^{-1}	Fujiwara et al 2007
Probability of nucleating a new filament	7.51 x 10^{-16}	ms^{-1}	Belzner & Pollard 2008
Probability of bundling protein binding	0.1	ms^{-1}	
Probability of bunding protein dissociating	0.01	ms^{-1}	

There is no observable trend in the data for either simulation. The 10-fold difference in curvature did not affect the overall shape of the graph.

Output from the simulation shows that the bending events for the curvature of 0.001 nm^{-1} were relatively frequent, but as the curvature is small, the difference in length for each event was negligible. The simulations with 0.01nm^{-1} curvature showed less frequent bending events, with higher reductions in length.

4. Discussion

Fluorescence images of actin filaments show them taking on very bent conformations (Skau et al, 2011), while in this simulation they either only bend a small amount, or bend very infrequently.

Only the probability of a filament bending to a particular discrete curvature is calculated in this simulation. In reality, bending is a continuous process. A filament would first bend a small amount, and then further, rather than simply switching between two conformations. This discrepancy could account for the lack of any obvious trend in my results.

A more accurate way of simulating bending would be to find a probability distribution, over the full range of curvatures. A random number could then determine the curvature to which the filament will bend.

4.1. Future directions

One other significant improvement to the simulation would be reducing the time it takes to run. Many of the calculations could be performed in parallel, and the speed of each step could possibly be increased by using Python modules such as “NumPy” and “SciPy”. This would allow for runs with more realistic numbers of filaments.

Currently, the filament and bundling protein agents each update in sequence. This means that the first agent in each group will do more work over the course of the simulation. A possible fix for this would be to randomise the order of updating.

This simulation is one dimensional, meaning that only the length of the membrane invagination is measured. An interesting future project could be to expand the simulation to two or even three dimensions. This way, the geometry of the membrane invagination could be examined. This type of simulation could be used to ask why the polymerising actin pulls the tip of the invagination inwards rather than pushing the outer membrane outwards.

Since the growth of branched filaments, nucleated by Arp2/3 is considered to be an important part of endocytosis, a two or three dimensional simulation could consider the forces produced when the filament pushing the membrane is not at a 90 degree angle, or when there are multiple branches.

Endocytosis is a complex process, that involves many different proteins that interact with actin, and with the cell membrane. The influence of capping and severing of actin filaments could be explored, as well as the action of proteins that deform the membrane.

4.2. Conclusion

I have created a one dimensional, agent-based Monte Carlo simulation of actin filaments that shows polymerisation, depolymerisation, bundling and bending, as well as movement of a tethered membrane. Program output shows that the filaments grow to reach the outer membrane, at which point they stop growing, unless one of them pushes away from the outer membrane.

The simulation outputs data files, graphs, snapshots, and animations of the process, and can be used to find the amount of time it takes the invagination reach the required length for successful endocytosis, or to track the events in a single run.

In the future, this program could be used to investigate the roles of various players in the process of endocytosis. These players currently include bundling proteins, membrane tension, Young's modulus, G-actin concentration, F-actin concentration and temperature. If the simulation were expanded, capping proteins, severing proteins, branching and membrane deformation could also be examined using this program.

Acknowledgements

With thanks to Rhoda Hawkins for her guidance and supervision (particularly with the maths!), as well as Kathryn Ayscough for insights into the biological background of yeast endocytosis, and Jeremy Craven for his invaluable Python classes.

5. References

Abd, T., & F-actin, T. (1998). letters An atomic model of fimbrin binding to F-actin and its implications for filament crosslinking and regulation, 5(9), 1–6.

Allain, J.-M., Storm, C., Roux, a., Amar, M., & Joanny, J.-F. (2004). Fission of a Multiphase Membrane Tube. Physical Review Letters, 93(15), 1–4. doi:10.1103/PhysRevLett.93.158104

Baumgart, T., Hess, S. T., & Webb, W. W. (2003). Imaging coexisting fluid domains in biomembrane models coupling curvature and line tension. Nature, 425(6960), 821–4. doi:10.1038/nature02013

Beltzner, C. C., & Pollard, T. D. (2008). Pathway of actin filament branch formation by Arp2/3 complex. The Journal of biological chemistry, 283(11), 7135–44. doi:10.1074/jbc.M705894200

Berro, J., Sirotkin, V., & Pollard, T. D. (2010). Mathematical Modeling of Endocytic Actin Patch Kinetics in Fission Yeast : Disassembly Requires Release of Actin Filament Fragments, 21, 2905–2915. doi:10.1091/mbc.E10

Brangwynne, C. P., Koenderink, G. H., Barry, E., Dogic, Z., MacKintosh, F. C., & Weitz, D. a. (2007). Bending dynamics of fluctuating biopolymers probed by automated high-resolution filament tracking. Biophysical journal, 93(1), 346–59. doi:10.1529/biophysj.106.096966

Cheng, D., Marner, J., & Rubenstein, P. a. (1999). Interaction in vivo and in vitro between the yeast fimbrin, SAC6P, and a polymerization-defective yeast actin (V266G and L267G). The Journal of biological chemistry, 274(50), 35873–80. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/10585472

Dominguez, R., & Holmes, K. C. (2011). Actin structure and function. Annual review of biophysics, 40, 169–86. doi:10.1146/annurev-biophys-042910-155359

Fujiwara, I., Vavylonis, D., & Pollard, T. D. (2007). Polymerization kinetics of ADP- and ADP-Pi-actin determined by fluorescence microscopy. Proceedings of the National Academy of Sciences of the United States of America, 104(21), 8827–32. doi:10.1073/pnas.0702510104

Hawkins, R., Piel, M., Faure-Andre, G., Lennon-Dumenil, a., Joanny, J., Prost, J., & Voituriez, R. (2009). Pushing off the Walls: A Mechanism of Cell Motility in Confinement. Physical Review Letters, 102(5), 1–4. doi:10.1103/PhysRevLett.102.058103

Idrissi, F.-Z., Grötsch, H., Fernández-Golbano, I. M., Presciatto-Baschong, C., Riezman, H., & Geli, M.-I. (2008). Distinct acto/myosin-I structures associate with endocytic profiles at the plasma membrane. The Journal of cell biology, 180(6), 1219–32. doi:10.1083/jcb.200708060

Kaksonen, M. (2008). Taking apart the endocytic machinery. The Journal of cell biology, 180(6), 1059–60. doi:10.1083/jcb.200802174

Kaksonen, M., Sun, Y., & Drubin, D. G. (2003). A pathway for association of receptors, adaptors, and actin during endocytic internalization. Cell, 115(4), 475–87. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/14622601

Kaksonen, M., Toret, C. P., & Drubin, D. G. (2005). A modular design for the clathrin- and actin-mediated endocytosis machinery. Cell, 123(2), 305–20. doi:10.1016/j.cell.2005.09.024

Kojima, H., Ishijima, a, & Yanagida, T. (1994). Direct measurement of stiffness of single actin filaments with and without tropomyosin by in vitro nanomanipulation. Proceedings

of the National Academy of Sciences of the United States of America, 91(26), 12962–6. Retrieved from http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=45560&tool=pmcentrez&rendertype=abstract

Li, R. (1997). Bee1, a yeast protein with homology to Wiscott-Aldrich syndrome protein, is critical for the assembly of cortical actin cytoskeleton. The Journal of cell biology, 136(3), 649–58. Retrieved from http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2134298&tool=pmcentrez&rendertype=abstract

Liu, J., Kaksonen, M., Drubin, D. G., & Oster, G. (2006). Endocytic vesicle scission by lipid phase boundary forces. Proceedings of the National Academy of Sciences of the United States of America, 103(27), 10277–82. doi:10.1073/pnas.0601045103

Liu, J., Sun, Y., Drubin, D. G., & Oster, G. F. (2009). The mechanochemistry of endocytosis. PLoS biology, 7(9), e1000204. doi:10.1371/journal.pbio.1000204

Naqvi, S. N., Zahn, R., Mitchell, D. a, Stevenson, B. J., & Munn, a L. (1998). The WASp homologue Las17p functions with the WIP homologue End5p/verprolin and is essential for endocytosis in yeast. Current biology : CB, 8(17), 959–62. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/9742397

Norman, A. I., Ivkov, R., Forbes, J. G., & Greer, S. C. (2005). The polymerization of actin: structural changes from small-angle neutron scattering. The Journal of chemical physics, 123(15), 154904. doi:10.1063/1.2039088

Pollard, T. D., & Berro, J. (2009). Mathematical models and simulations of cellular processes based on actin filaments. The Journal of biological chemistry, 284(9), 5433–7. doi:10.1074/jbc.R800043200

Pollard, T. D., & Borisy, G. G. (2003). Cellular motility driven by assembly and disassembly of actin filaments. Cell, 112(4), 453–65. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/12600310

Pollard, T. D., & Cooper, J. a. (2009). Actin, a central player in cell shape and movement. Science (New York, N.Y.), 326(5957), 1208–12. doi:10.1126/science.1175862

Roux, A., Cuvelier, D., Nassoy, P., Prost, J., Bassereau, P., & Goud, B. (2005). Role of curvature and phase transition in lipid sorting and fission of membrane tubules. The EMBO journal, 24(8), 1537–45. doi:10.1038/sj.emboj.7600631

Sept, D., & McCammon, J. a. (2001). Thermodynamics and kinetics of actin filament nucleation. Biophysical journal, 81(2), 667–74. doi:10.1016/S0006-3495(01)75731-1

Shih, S. C., Katzmann, D. J., Schnell, J. D., Sutanto, M., Emr, S. D., & Hicke, L. (2002). Epsins and Vps27p/Hrs contain ubiquitin-binding domains that function in receptor endocytosis. Nature cell biology, 4(5), 389–93. doi:10.1038/ncb790

Sirotkin, V., Berro, J., Macmillan, K., Zhao, L., & Pollard, T. D. (2010). Quantitative Analysis of the Mechanism of Endocytic Actin Patch Assembly and Disassembly in Fission Yeast, 21, 2894–2904. doi:10.1091/mbc.E10

Skau, C. T., Courson, D. S., Bestul, A. J., Winkelman, J. D., Rock, R. S., Sirotkin, V., & Kovar, D. R. (2011). Actin filament bundling by fimbrin is important for endocytosis, cytokinesis, and polarization in fission yeast. The Journal of biological chemistry, 286(30), 26964–77. doi:10.1074/jbc.M111.239004

Smith, M. G., Swamy, S. R., & Pon, L. a. (2001). The life cycle of actin patches in mating yeast. Journal of cell science, 114(Pt 8), 1505–13. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/11282026

Smythe, E., & Ayscough, K. R. (2006). Actin regulation in endocytosis. Journal of cell science, 119(Pt 22), 4589–98. doi:10.1242/jcs.03247

Stricker, J., Falzone, T., & Gardel, M. L. (2010). Mechanics of the F-actin cytoskeleton. Journal of biomechanics, 43(1), 9–14. doi:10.1016/j.jbiomech.2009.09.003

Theriot, J. a. (1997). Accelerating on a treadmill: ADF/cofilin promotes rapid actin filament turnover in the dynamic cytoskeleton. The Journal of cell biology, 136(6), 1165–8. Retrieved from http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2132515&tool=pmcentrez&rendertype=abstract

6. Appendices

6.1. Code

6.1.1. Main program: main.py

This program imports the functions, parameters and classes from the other modules and runs them in the right order

```
#! /usr/bin/python
# takes number of iterations (tmax in ms) as optional command line argument

import random, sys, math
from parameters import *
from functions import *
from classes import *

files = openfiles()

# creates bundling protein dictionary, which is full of bundling protein objects
bundlingregister = {}
for i in range(0,bundlingnumber):
	bundlingregister[i] = bundlingptn(i)

# creates filament dictionary, which is full of filament objects
filaments = {}
for i in range(0,numberoffilaments):
	filaments[i] = filament(i)

# creates membrane object
membrane = membrane(barrier)

# empty lists for recording the longest filament, and the number of active filaments
longestfilament = [1]
activefils = [numberoffilaments]
touchingbarrier = [0]

# growing filaments and storing list of lengths in values dictionary with filament no as
key
time = 0
while time < 100:
#while membrane.position < 40:

	# count the total number of filaments
	numberoffilaments = count(filaments)

	# record the total number of active filaments
```

```
        activefils = count(filaments,"active")

        # record the length of the longest filament
        longestfilament = longest(filaments)

        # count number of filaments touching the barrier
        touchingbarrier = touching(filaments,membrane)

        # add and remove bundling proteins
        bundlingregister = bundles(bundlingregister,filaments)

        # pass information between bundling proteins and filaments
        filaments = bundlemessage(bundlingregister,filaments)

        # grow, bend or shrink filaments
        filaments = growing(filaments,touchingbarrier, membrane)

        # push against the membrane
        membrane.moveon(longest(filaments))

        # nucleate new filaments
        filaments = nucleate(filaments,time)

        # output data

output(time,activefils,longestfilament,filaments,membrane,touchingbarrier,bundlingregi
ster,files)

        # increment time
        time+=dt
```

6.1.2. Repeatedmain.py

This is a variation on the previous program, which allows the simulation to be called repeatedly with different parameters each time.

```
#! /usr/bin/python
# takes number of iterations (tmax in ms) as optional command line argument

import random, sys, math
from parameters import *
from functions import *
from classes import *

mafile = open('output/'+str(y),'w+')
metaoutput = []
z = membranetension

for y in range(0,60):
 for x in range(1,20):
  files = openfiles()

 # creates bundling protein dictionary, which is full of bundling protein objects
  bundlingregister = {}
  for i in range(0,bundlingnumber):
        bundlingregister[i] = bundlingptn(i)

 # creates filament dictionary, which is full of filament objects
  filaments = {}
  for i in range(0,numberoffilaments):
        filaments[i] = filament(i)

 # creates membrane object
  membrane = membraney(barrier)

 # empty lists for recording the longest filament, and the number of active filaments
  longestfilament = [1]
  activefils = [numberoffilaments]
  touchingbarrier = [0]

 # growing filaments and storing list of lengths in values dictionary with filament no as
key
  time = 0
  while membrane.position < 40:

        # count the total number of filaments
        numberoffilaments = count(filaments)
```

```
        # record the total number of active filaments
        activefils = count(filaments,"active")

        # record the length of the longest filament
        longestfilament = longest(filaments)

        # count number of filaments touching the barrier
        touchingbarrier = touching(filaments,membrane)

        # add and remove bundling proteins
        bundlingregister = bundles(bundlingregister,filaments)

        # pass information between bundling proteins and filaments
        filaments = bundlemessage(bundlingregister,filaments)

        # grow, bend or shrink filaments
        filaments = growing(filaments,touchingbarrier, membrane)

        # push against the membrane
        membrane.moveon(longest(filaments))

        # nucleate new filaments
        filaments = nucleate(filaments,time)

output(time,activefils,longestfilament,filaments,membrane,touchingbarrier,bundlingregi
ster,files)

        # increment time
        time+=dt
  string = str(z) + "\t" + str(y) + "\t" + str(time) + "\n"
  metaoutput.append([z,y,time])
  print string
  mafile.write(string)
```

6.1.3. Classes.py

This module defines the three classes of agent.

```
# classes of object
from parameters import *

class membrane:
	"""The cell membrane, with position and history"""
	hardness = 10
	def __init__(self, number):
		self.position = number
	def moveon(self, longestfilament):
		"""Change the position if pushed and increase hardness"""
		if self.position < longestfilament:
			newposition = longestfilament
			distance = longestfilament - self.position
			self.hardness += (membranetension * distance)
		else:
			newposition = self.position
		self.position = newposition

class filament:
	"""A single actin filament, with length, id and bundling proteins"""
	length = 3*unitlength
	bent = 0.0
	bundled = "alone"
	radiusofbundle = []
	def __init__(self, number):
		self.n = number
	def moveon(self, newlength):
		"""Change the length """
		self.length = newlength
		return None
	def bundler(self, newthing):
		"""Add or remove bundling proteins"""
		self.bundled = newthing
		return None

class bundlingptn:
	"""A single bundling protein"""
	def __init__(self, number):
		self.n = number
	position = "a"
	attachedto = "n"
	def moveon(self, newpos, pair):
		""" Change the position and pairing """
```

```
                self.position = newpos
                self.attachedto = pair
```

6.1.4. Parameters.py

This module contains all the parameters for the program.

```
#! /usr/bin/python
import sys

# probabilities sum to 1
# from fujiwara 2007 and sirotkin 2010
# in time step 1ms and [G-actin] 22uM
probability_addone = 0.25520
probability_minusone = 0.00025
probability_staysame = 0.74455
# from Beltzner & Pollard 2008
# 8 x 10^-9 uM-3 s-1
# weird units: divide by concentration of G-actin cubed
# 7.51 x 10^-16 ms-1
probability_nucleation = 7.51*10**-16
probability_bend = 0.1
probability_bundle = 0.1
probability_unbundle = 0.01

# From Liu et al 2006
# 10^-4 N/m
# units N/nm
membranetension = 10**-13

# number of filaments
numberoffilaments = 10

# number of bundling proteins
bundlingnumber = 10
#bundlingnumber = int(numberoffilaments/8)
#time step and tmax, time is measured in ms
dt = 1
try:
        tmax = int(sys.argv[1])
except:
        tmax = 100
# in nm
unitlength = 2.7
barrier = unitlength * 7
youngsmodulus_pernm = 1.8*(10**(-9))
# kelvin
temperature = 298
boltzmann = 1.38*(10**(-14))
```

6.1.5. Functions.py

This module defines all the functions used by the main program.

```
#! /usr/bin/python

from parameters import *
from classes import *
import decimal
import random, math, sys

# functions

def growing(filaments,touching,membrane):
	""" Grow, shrink and bend filaments """
	longestsofar = 0
	localbarrier = membrane.position
	amountpushed = membrane.position - barrier
	for individual in filaments:
		# takes the last item in the list of lengths
		thislength = filaments[individual].length
		# checks if the filament has disappeared
		if thislength > 0:
			# unbend if enough space
			bent = filaments[individual].bent
			if bent != 0:
				if thislength < localbarrier:
					if thislength+bent <= localbarrier:
						thislength += bent
						filaments[individual].bent = 0
					else:
						dist = localbarrier-thislength
						thislength += dist
						filaments[individual].bent -= dist
			# add or subtract subunit
			diceroll = random.random()
			if diceroll <= probability_addone:
				# checks to see if the filament hits the barrier
				if thislength+unitlength <= localbarrier:
					# adds a monomer if not
					thislength+=unitlength
				else:
					roulette = random.random()
					# push the barrier forwards
					probability_push = 2*math.exp((-
membranetension*amountpushed**2)/(boltzmann*temperature))/
(math.sqrt(boltzmann*temperature*math.pi/membranetension))
```

```
                                        if roulette < probability_push:
                                                thislength+=unitlength
                                                localbarrier+=unitlength
                                        else:
                                        # bend as a single filament
                                                filaments[individual] =
bend(filaments[individual],localbarrier,amountpushed,"single")
                                        # bend as a group
                                                bendingmessage =
bend(filaments[individual],localbarrier,amountpushed,"group")
                                                if bendingmessage[1] != 0:
                                                        for o in range(0,len(filaments)):
                                                                if o in bendingmessage[0]:
                                                                        #print "before", o,
filaments[o].bent, filaments[o].length
                                                                        filaments[o].bent +=
bendingmessage[1]
                                                                        filaments[o].length -=
bendingmessage[1]
                                                                        #print "after", o,
filaments[o].bent, filaments[o].length
                                # remove a monomer
                                elif diceroll <= probability_addone+probability_minusone:
                                        thislength-=unitlength

                        # add new length to the filament object
                        filaments[individual].moveon(thislength)
        return filaments

def count(filaments,*param):
        """ Count the number of (active or total) filaments """
        counter = 0
        for item in filaments:
                if param == "active":
                        if filaments[item].length > 0:
                                counter += 1
                else:
                        counter+=1
        return counter

def touching(filaments,membrane):
        counter = 0
        for item in filaments:
                if filaments[item].length == membrane.position:
                        counter += 1
        return counter
```

```
def longest(filaments):
	""" Find the length of the longest filament """
	longestsofar = 0
	for item in filaments:
		if filaments[item].length > longestsofar:
			longestsofar = filaments[item].length
	return longestsofar

def nucleate(filaments,time):
	""" Nucleate new filaments """
	nofils = numberoffilaments
	nucleationdiceroll = random.random()
	if nucleationdiceroll <= probability_nucleation:
		filaments[nofils] = filament(numberoffilaments+1)
		filaments[nofils].history = [0]*((time/dt))
		filaments[nofils].length = 1
	return filaments

def bundlemessage(bundlingregister,filaments):
	""" Passes messages between bundles and filaments """
	message = []
	wholebundle = []
	# creates a list with an entry for each filament
	# with one empty list (the position of bundling proteins)
	# and 1 for the radius of the bundle
	for item in filaments:
		message.append([[],1])
	for item in bundlingregister:
		# if the bundling protein is attached to two filaments
		if bundlingregister[item].position != 'a':
			pos = bundlingregister[item].position
			pal = bundlingregister[item].attachedto
			# put the position of the bundle into the message for each filament
			message[pal[0]][0].append(pos)
			message[pal[1]][0].append(pos)
			# makes lists of bundles of filaments by stepping through each
bundling protein
			nowhere = True
			for abc in wholebundle:
				if pal[0] in abc:
					nowhere = False
					if pal[1] not in abc:
						abc.append(pal[1])
				elif pal[1] in abc:
					nowhere = False
```

```
                                        abc.append(pal[0])
                    # makes a new bundle list if the filament is not in another bundle
                    if nowhere:
                        wholebundle.append([pal[0],pal[1]])
    wholebundle = condense(wholebundle)
    # checks to see if each filament is attached to something or not
    # gives back a bundle size for each filament
    for i in range(0,len(message)):
        if message[i][0] == []:
            message[i][0] = "alone"
        bsize = [i]
        for thing in wholebundle:
            if thing != None:
                if i in thing:
                    bsize = thing
        message[i][1] = bsize
    # adds the message to the filament objects
    for individual in filaments:
        filaments[individual].bundler(message[individual][0])
        filaments[individual].radiusofbundle = message[individual][1]
    return filaments

def bundles(bundlingregister,filaments):
    """ Give bundling proteins a position """
    for item in bundlingregister:
        bundleroll = random.random()
        if bundlingregister[item].position == 'a':
            if bundleroll <= probability_bundle:
                bundlingregister[item].attached = 1
                filist = []
                for i in range(0,numberoffilaments):
                    filist.append(i)
                first = random.choice(filist)
                filist.remove(first)
                second = random.choice(filist)
                attachedto = [first,second]
                limit = min(filaments[first].length,filaments[second].length)
                newposition = random.randint(1,math.floor(limit))
            else:
                attachedto = "n"
                newposition = "a"
        else:
            if bundleroll <= probability_unbundle:
                bundlingregister[item].attached = 0
                newposition = "a"
                attachedto = "n"
```

```
                        else:
                                newposition = bundlingregister[item].position
                                attachedto = bundlingregister[item].attachedto
                bundlingregister[item].moveon(newposition,attachedto)
        return bundlingregister

def condense(wholebundle):
        ''' combine bundles that share common filaments'''
        oldgroup = []
        if len(wholebundle) > 0:
                for i in range(0,len(wholebundle)):
                        condition = False
                        group = wholebundle[i]
                        for item in oldgroup:
                                if item in group:
                                        wholebundle[i].extend(oldgroup)
                                        wholebundle[i] = list(set(wholebundle[i]))
                                        wholebundle[i-1] = None
                        oldgroup = group
                for thing in wholebundle:
                        if thing == None:
                                wholebundle.remove(thing)
        return wholebundle

def bend(filament,localbarrier,amountpushed,*param):
        shorten = 0
        if "single" in param:
                groupradius = 3.5
                if filament.bundled != "alone":
                        grouplength = localbarrier - int(max(filament.bundled))
                else:
                        grouplength = filament.length
                gp = False
        elif "group" in param:
                groupradius = len(filament.radiusofbundle)*3.5
                grouplength = filament.length
                gp = True

        curvature = 0.001
        mominert = (math.pi * groupradius **4) /4
        fx = amountpushed * membranetension * unitlength
        groupbend = (fx - (curvature**2) * youngsmodulus_pernm * grouplength *
mominert )/ (boltzmann * temperature)
```

```
	scalingfactor = 1/(math.exp(fx / (boltzmann * temperature)) *
(math.sqrt(boltzmann * temperature * math.pi))/(2*math.sqrt(youngsmodulus_pernm *
grouplength * mominert)))

#	scalingfactor = (youngsmodulus_pernm * curvature**2 * mominert)/(boltzmann *
temperature)
#	scalingfactor = 2*(math.sqrt((youngsmodulus_pernm * grouplength * mominert /
(boltzmann * temperature * math.pi)))

	probability_groupbend = scalingfactor*math.exp(groupbend)
	#print "P=", probability_groupbend, "N=", scalingfactor, "F=",
amountpushed*membranetension, "x=", unitlength, "Y=", youngsmodulus_pernm, "l=",
grouplength, "I=", mominert, "kB=", boltzmann, "T=", temperature

	if random.random() < probability_groupbend:
		shorten = grouplength-(1/curvature)*math.sin(grouplength*curvature)
		if not gp:
			filament.bent += shorten
			filament.length -= shorten
	if gp:
		return [filament.radiusofbundle,shorten]
	else:
		return filament

def
output(time,activefils,longestfilament,filaments,membrane,touchingbarrier,bundlingregi
ster,files):
 """writing data out to files"""

 affile = files[0]
 lefile = files[1]
 onfile = files[2]
 bnfile = files[3]
 bpfile = files[4]
 befile = files[5]

 string = str(time) + "\t" + str(activefils) + "\n"
 affile.write(string)

 string = str(time) + "\t" + str(longestfilament) + "\n"
 onfile.write(string)

 string = str(time) + "\t"
 for number in range(0,numberoffilaments):
	try:
		string += str(filaments[number].length)+"\t"
```

```
        except:
                string += "\t"
string+= "\n"
lefile.write(string)

string = str(time) + "\t"
for number in range(0,numberoffilaments):
        try:
                string += str(filaments[number].bent)+"\t"
        except:
                string += "\t"
string+= "\n"
befile.write(string)

string = str(time) + "\t"
for thing in bundlingregister:
                place = bundlingregister[thing].position
                if place != "a":
                        first = bundlingregister[thing].attachedto[0]
                        second = bundlingregister[thing].attachedto[1]
                else:
                        first = "n"
                        second = "n"
                string += str(place)+ "\t" + str(first) + "\t" + str(second) + "\t"
string += "\n"
bpfile.write(string)

string = str(time) + "\t" + str(membrane.position)+ "\t" + str(touchingbarrier) + "\n"
bnfile.write(string)

def openfiles():

        affile = open('output/output-number','w')
        lefile = open('output/output-length','w')
        onfile = open('output/output-longest','w')
        bnfile = open('output/output-barriernumber','w')
        bpfile = open('output/output-bundling','w')
        befile = open('output/output-bent','w')

        return [affile,lefile,onfile,bnfile,bpfile,befile]
```

6.1.6. Graphs.py

This program is to be run after the main program. It takes the output files and creates graphs.

```
#! /usr/bin/python

from pyx import *

graphsize = 50

#makes line graph of number of active filaments
g = graph.graphxy(width=graphsize,
        x=graph.axis.lin(title="Time (s)"),
        y=graph.axis.lin(min=0, title="Number of filaments"))
g.plot(graph.data.file("output/output-number", x=1, y=2),
        [graph.style.line()])
g.writeEPSfile("output/graph-number")

#counts number of filaments in the length file
lf = open('output/output-length','r')
columns = len(lf.readline().split())

#makes line graph of length vs time, with longest filament in red
p = graph.graphxy(width=graphsize,
     x=graph.axis.lin(title="Time (ms)"),
     y=graph.axis.lin(title="Length"))
for i in range(2,columns):
        p.plot([graph.data.file("output/output-length", x=1, y=i)],[graph.style.line()])
p.plot([graph.data.file("output/output-longest", x=1, y=2)],
[graph.style.line([color.rgb.red])])
p.writeEPSfile("output/graph-length")

q = graph.graphxy(width=graphsize,
        x=graph.axis.lin(title="Time(ms)"),
        y=graph.axis.lin(),
        key=graph.key.key(pos="tl", dist=0.1))
q.plot(graph.data.file("output/output-barriernumber", x=1, y=2, title="Barrier position"),
[graph.style.line()])
q.plot(graph.data.file("output/output-barriernumber", x=1, y=3, title="Number of
filaments touching barrier"),[graph.style.line([color.rgb.red])])

q.writeEPSfile("output/graph-barriernumber")
```

6.1.7 Images.py

This program is to be run after the main program. It takes the output files and creates postscript images.

```
#! /usr/bin/python

from pyx import *

barrier = []
barrierfile = open('output/output-barriernumber')
for item in barrierfile:
	item = item.split()
	barrier.append(float(item[1]))

filaments = []
filamentsfile = open('output/output-length')
for item in filamentsfile:
	item = item.split()
	filaments.append(item[1:])

bundling = []
bundlingfile = open('output/output-bundling')
for item in bundlingfile:
	item = item.split()
	bundling.append(item[1:])
noofbuns = len(bundling[0])/3

nooffils = len(filaments[0])
boundary = max(barrier)

universeacross = path.line(0,0,float(boundary)+2,0)
universeup = path.line(0,0,0,float(nooffils)+3)
backmembrane = path.line(1, 1, 1, nooffils+2)

tmax = len(barrier)

for i in range(0,tmax):
	tether = barrier[i]+1
	frontmembrane = path.line(tether,1,tether,nooffils+2)
	c = canvas.canvas()
	c.stroke(universeup,[color.rgb.white])
	c.stroke(universeacross,[color.rgb.white])
	index = 2
	for item in filaments[i]:
		start = tether - float(item)
		statement = path.line(start,index,tether,index)
```

```
                index += 1
                c.stroke(statement,[style.linewidth.THICK,color.rgb.blue])
        for a in range(0,noofbuns):
                position = bundling[i][3*a]
                if position != "a":
                        start = tether - float(position)
                        first = float(bundling[i][3*a+1])+2
                        second = float(bundling[i][3*a+2])+2
                        bundleone = path.circle(start,first,0.1)
                        bundletwo = path.circle(start,second,0.1)
                        connector = path.line(start,first,start,second)
                        c.stroke(connector)
                        c.fill(bundleone)
                        c.fill(bundletwo)

        c.stroke(backmembrane,[style.linewidth.THICK,color.rgb.red])
        c.stroke(frontmembrane,[style.linewidth.THICK,color.rgb.red])

        filename = '{0:04}'.format(i)
        c.writeEPSfile('output/'+filename)
```

6.1.7. Animation.sh

This bash script can be used to create an animated .gif from the postscript files created by images.py. It uses the ImageMagick command line tools. Since this uses up a significant amount of RAM, it is not particularly useful for simulations over 100ms long.

```
#! /bin/bash

convert -delay 20 -dispose Previous -alpha opaque -size 10000 ./output/????.eps
./output/animation.gif
```

6.1.8. Newimages.py

This program is to be run after the main program. It takes the output files and creates an animation using the PyGame module.

```
#! /usr/bin/python

import pygame, sys, math

def cv(number):
	number = (number*40)+40
	return number

def ch(number):
	number = (number*20)+20
	return number

pygame.init()

bent = []
bentfile = open('output/output-bent')
for item in bentfile:
	item = item.split()
	bent.append(item[1:])

barrier = []
barrierfile = open('output/output-barriernumber')
for item in barrierfile:
	item = item.split()
	barrier.append(float(item[1]))

filaments = []
filamentsfile = open('output/output-length')
for item in filamentsfile:
	item = item.split()
	filaments.append(item[1:])

bundling = []
bundlingfile = open('output/output-bundling')
for item in bundlingfile:
	item = item.split()
	bundling.append(item[1:])
noofbuns = len(bundling[0])/3

nooffils = int(len(filaments[0]))
boundary = int(math.ceil(max(barrier)))
```

```
size = width, height = ch(boundary)+20, cv(nooffils)+20
red = 255, 0, 0
blue = 0, 0, 255
black = 0,0,0
white = 255,255,255

screen = pygame.display.set_mode(size)

tmax = len(barrier)

while 1:
	screen.fill(black)
	for i in range(0,tmax):
		# allow exit
		for event in pygame.event.get():
			if event.type == pygame.QUIT: sys.exit()
		# draw leftmost membrane
		pygame.draw.line(screen, red, [20, 20], [20, cv(nooffils)], 3)
		# draw rightmost membrane
		pygame.draw.line(screen, red, [ch(barrier[i]),20],
[ch(barrier[i]),cv(nooffils)],3)
		x = 40
		for item in range(0,len(filaments[i])):
			length = float(filaments[i][item])
			start = ch(barrier[i]) - length*20
			pygame.draw.line(screen, blue, [start,x], [ch(barrier[i]),x],3)
			vertical = float(bent[i][item])
			#print vertical
			curve = pygame.Rect(start-5, x, (ch(barrier[i])-start), vertical)
#			pygame.draw.arc(screen,blue,curve,math.pi/2, math.pi)
			x += 40
		for a in range(0,noofbuns):
			if bundling[i][3*a] != "a":
				distance = ch(int(barrier[i])) - int(10*(float(bundling[i][3*a])))
				first = cv(int(bundling[i][3*a+1]))
				second = cv(int(bundling[i][3*a+2]))
				crcolor = 255,255,int(a)*25
				pygame.draw.circle(screen,crcolor ,[distance,first], 3)
				pygame.draw.circle(screen,crcolor ,[distance,second], 3)
				pygame.draw.line(screen,crcolor ,[distance,first],
[distance,second],3)
		pygame.display.flip()
		pygame.time.delay(300)
		screen.fill(black)
```