Curvature Mediated Interactions Documentation

Lucas Johnson *lucasjoh@usc.edu*

July 2017

Contents

1 Introduction

1.1 Project Description (from Dr. Haselwandter)

Membrane shape regulation is essential for all cellular membrane trafficking events as well as the formation and maintenance of all cellular organelles. A series of seminal discoveries made a decade ago revealed [see attached article by McMahon *et al.*] that membrane curvature generation by certain peripheral membrane proteins plays a central role in membrane shape regulation. While a number of molecular mechanisms for curvature generation have been identified, even some of the most basic aspects of how bilayer-protein interactions produce the observed isotropic and anisotropic membrane curvatures remain unknown. A fundamental obstacle limiting our understanding of membrane remodeling is that membrane shape regulation cannot be understood by considering only individual proteins. Instead, membrane remodeling is a collective phenomenon that depends on protein concentration as well as the clustering and, to generate anisotropic curvature, ordering of many interacting proteins.

The aim of this project is to develop a mathematical framework for the analytic and numerical study of curvature-mediated interactions between membrane proteins. Such a framework can be formulated based on continuum elasticity theory which, dating back to the seminal work of W. Helfrich, P. B. Canham, and E. A. Evans, has provided much insight into the physical mechanisms controlling membrane shape [see, e.g., D. H. Boal, *Mechanics of the Cell* (Cambridge University Press, 2012) and Sec. 1 of the attached notes]. From a mathematical perspective, the basic task here is to minimize the so-called Helfrich-Canham-Evans energy for elastic lipid bilayer membranes which are deformed by two (or more) membrane proteins. Membrane proteins are generally rigid compared to the surrounding lipid bilayer, and can therefore be regarded as imposing fixed boundary conditions on the elastic surface provided by lipid bilayers (and described by the Helfrich-Canham-Evans energy). I suggest that we tackle this problem as follows:

1) Computational approach: The numerical calculation of curvature-mediated interactions between membrane proteins presents considerable mathematical difficulties. Recent work suggests that the finite element software package *Surface Evolver* [1] allows numerical calculation of curvature-mediated interactions between isotropic (conical) [see attached article by Reynwar *et al.*] as well as anisotropic [see attached article by Schweitzer *et al.*] membrane proteins. However, the set-up and use of this software package appears to be far from straightforward. A first task will therefore be to set-up *Surface Evolver* for the calculation of curvature-mediated interactions, and to thoroughly check this set-up for isotropic and anisotropic membrane proteins against the results of Reynwar *et al.* and Schweitzer *et al.*

2) Analytic approach: It is unclear to what extent the solutions of curvature-mediated protein interactions obtained with the *Surface Evolver* package are mathematically accurate. We have shown previously how curvature-mediated interactions between two isotropic (conical) membrane proteins can be calculated analytically for two proteins at arbitrary separations [see Secs. 2.1 and 2.2 of attached notes], and a similar solution was also published recently [see attached paper by Fournier *et al.*]. I suggest that you work through these solutions and implement them in, e.g., *Mathematica.* We can then make a systematic comparison between the two approaches, and test the validity of the analytic and numerical solution schemes.

3) Anisotropic membrane proteins: With the numerical and analytic approaches both in place, we can tackle a number of interesting problems. In particular, following similar steps as in the attached paper by Haselwandter *et al.* (see also Sec. 3 of the attached notes), it should be feasible to generalize the analytic solution of curvature-mediated protein interactions to proteins with a circular cross section with a bilayer-protein contact angle which varies along the bilayer-protein interface, as well as proteins which do not have a circular cross section. It would be exciting to develop such a novel analytic approach, and check it using *Surface Evolver.*

4) Pairwise additivity: It has been postulated for a long time that curvature-mediated protein interactions are not pairwise additive. Using the above combined analytic and numerical framework, we would be in a position to systematically investigate pairwise additivity of curvature-mediated protein interactions through direct analytic and numerical calculation. This would offer direct insights into curvature-mediated interactions among many membrane proteins in close proximity, which is the scenario most relevant for membrane shape regulation.

1.2 Current State of Project

At the time of writing this, all files and scripts mentioned are uploaded to github at `https://github.com/lucdj3/Haselwandter-Lab/tree/master/LabStuff` in their most up to date form. Surface Evolver and Gmsh are themselves not in this repository, but must be downloaded for use of nearly everything in the repository. ***The project has been carried out on a computer equipped with Windows 10 running the Creator's Update, executing commands via 'Bash on Ubuntu on Windows' available through this update. Many parts of the python and c++ scripts may need modification to function properly in other environments.***

Presently, the use of a method devised by Osman and me, which I will refer to as the "Osman-Lucas Method", seems to be the most promising. The method involves building a mesh in *Gmsh* [2], applying necessary changes to the file, and importing it to *Surface Evolver* to find the minimum energy shape and energy value. This method has provided highly accurate results for isotropic single inclusion (conical) models on circular membranes of varying size, using equation (6) presented in Auth *et al.* for analytic comparison. As one would expect, obtaining accurate results from *Surface Evolver* is highly dependent on the mesh produced by *Gmsh*, but unfortunately there are not clear qualities of a *Gmsh* script that make for accurate results; creation of the mesh involves more

[1]`http://facstaff.susqu.edu/brakke/evolver/evolver.html`

[2]`http://gmsh.info/`

trial and error than would be ideal, that is any at all. Further worrisome is that on occasion, I have produced meshes with significantly (>5%) lower energy in *Surface Evolver* than should correspond to such a mesh according to Auth *et al.*, even using a 1° contact angle which minimizes error due to estimates from Auth *et al.* The issue of finding too low of an energy is recent and currently a loose end.

Before finding the aforementioned point of worry, I began to apply a variation of this method to isotropic double inclusion models, like those studied in Reynwar *et al.*, with hopeful but presently inadequate results. The variation in method is due to a new degree of freedom, that of the polar tilt angle of the inclusion (in the single inclusion model, the polar axis will always be vertical). The new method simply involves running another script with the mesh produced by *Surface Evolver* via the initial method. The new script iterates many times over small changes in the polar tilt angle of the inclusion to find the tilt angle corresponding to lowest energy. An outer edge radius of 1280 (where the inclusion has a radius of 1), mirror symmetry such that only a semicircle and 1 inclusion must be used, varying distance of the inclusion center from the mirror plane, and varying contact angles of the inclusion with the membrane have been used for attaining minimized energies. Across a diversity of inclusion distances and contact angles, *Surface Evolver* has consistently calculated energy values between 10 and 20% **lower** than equation (33) of the attached notes from Christoph Haselwandter suggests is accurate – after doubling the energy from *Surface Evolver* as this energy is from only the semicircular model, half of the two inclusion system. Although I have not tried a wide variety of scripts for *Gmsh* for these two inclusion systems, I am confident that this is similar to the one inclusion system where highly accurate meshes can be produced, but such "accurate" meshes do not have clearly discernible qualities, making for a relatively unscientific procedure that is difficult to apply to models for which there are no analytic results to compare. A further obstacle for this method, although one that I believe is quite surmountable, is that models for which the minimum energy shape involves the inclusion tilting to a significant polar angle (>5°) start to create large faces around the inclusion in *Surface Evolver*, leading to poor accuracy of the energy calculated in the most important part of the mesh.

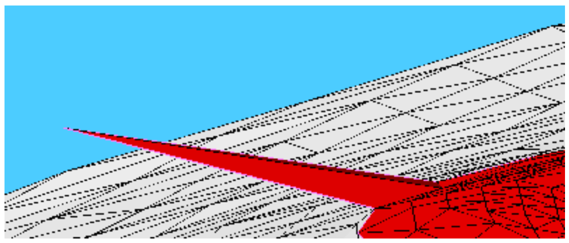
These descriptions of the most recent usage of the Osman-Lucas Method on isotropic single and double inclusion systems represent the current state of this project. Below I will discuss the initial approach, involving another potential method that became a dead-end for me, as well as specifics in regards to the Osman-Lucas Method.

2 Methods and their Future

2.1 Reynwar Method

What I will refer to as the "Reynwar Method" is the method used in the Reynwar *et al.* paper to successfully find the minimum-energy shape of isotropic double inclusion models using *Surface Evolver* as a computational approach. It is the method I feel is most promising, not only for the models examined in Reynwar *et al.*, but also for models involving more inclusions and anisotropic inclusions. Unfortunately I was unable to recreate this method as I will describe below and chose to explore other methods both because of necessity and to potentially have multiple methods for comparing results. From my understanding of the paper and its supplementary materials, this method uses only *Surface Evolver.* The paper and supplementary materials fairly adequately describe the use of this method except for some basic details of the initial mesh, specifically what the supplementary material means by "boundary constraint" (*Surface Evolver* has 'boundaries' and 'constraints') and how the initial mesh is formed. From getting familiar with *Surface Evolver*, I believe that the "boundary constraints" are what *Surface Evolver* calls 'level set constraints', which simply require an equation, such as $x = 0$, to which all vertices, edges, and facets with this constraint as a modifier will be confined. The *Surface Evolver* website does an adequate job of describing these constraints.

My attempts at this approach can be found in the 'ReynwarMethod' folder. The latest file, 'FixedProtein14.txt' seems to me a reasonable starting mesh for the approach, as the general shape of the mesh looks relatively as it should for inclusions of various distance, contact angle, and polar tilt angle after refinement and energy minimization. The energies remain much too high, however, because of how quickly the number of vertices, edges, and faces grows, preventing energy minimization in reasonable time with adequate mesh refinement in high-curvature areas. This is where the Reynwar Method becomes significant, and where I hit a dead-end. *Surface Evolver's* standard method for refining a mesh refines it equally everywhere, splitting every triangular face into 4 equal triangular faces, effectively halving each of the original face's edges. Most of the mesh does not need to be very refined, however, while other parts of the mesh, specifically around the inclusion and especially between the inclusion and mirror plane, require high levels of refinement to accurately portray the significant curvature. The Reynwar Method makes use of two key *Surface Evolver* functions to deal with this, namely *refine edge* and *delete edge*, to only refine and remove edges that satisfy certain requirements. When trying to refine specific edges, however, if the edge is on the border between inclusion and mesh, I consistently get behavior that renders the mesh unusable thereafter. Where the edge should have split and created a pair of new faces and edges relatively in the shape of the previous edge and face, the new edges and faces create a "spike" into or out of the inclusion, which does not change with energy-minimizing commands, as seen here:



To observe this, or a similar occurence, open 'FixedProtein14.txt' in *Surface Evolver*, minimize the energy with *hessian_seek* a few times. Then right click near the inclusion (densest part), press *shift+f*, then *z*, and drag your mouse towards the right to zoom in towards the inclusion. Repeat until you are adequately zoomed in, pressing *t* and then dragging the mesh as necessary if the mesh must be translated, and pressing *z* to zoom again (section **3.1.2** below may help with this). Once at a satisfactory view, right click on an edge on the border of the red and white area to find its edge number, and use the command *refine edges where id = x*, where *x* is the number previously found. This should recreate the error that has halted my progress with the Reynwar Method. I have spent a lot of time reading online about *Surface Evolver*, combing through Reynwar *et al.*, and playing around with *Surface Evolver* to overcome this issue, but to no avail. I tried to contact the creator of *Surface Evolver* with no luck. Perhaps contacting a member of Reynwar *et al.* could lead to progress.

2.2 Osman-Lucas Method

As described above, the Osman-Lucas Method uses not only *Surface Evolver*, but also *Gmsh* as well as python and c++ scripts to manipulate meshes outside of *Surface Evolver*, and only find minimum-energy shape and energy value within *Surface Evolver*. The folders '1_inclusionGmsh' and '2_inclusionGmsh' contain '.geo' files which *Gmsh* can use to make initial 2-dimensional meshes. These meshes are then put through a python script, 'vtk_poly.py', to be prepared for 'bc_flags', which both converts the text of the file such that it can be used with *Surface Evolver*, and implants the necessary constraints and changes to create the inclusion (in actuality, this is all managed by 'runall.py'). In the case of 1 inclusion systems, the energy minimization with *Surface Evolver* is all that remains. In the case of 2 inclusion systems which must look at minimized energy over variable polar angle tilts, the file prepared for *Surface Evolver* is put through 'tilt_convert.py', which finds the minimum energy shape for each polar angle tilt between that of the incoming file ($0°$) and that of the user's choice, using $0.05°$ changes (this is easily adjustable). For each of these polar angle tilts, the angle and corresponding energy are printed to an output file.

The idea of this method is to only manipulate the shape of the mesh in *Surface Evolver*, and use *Gmsh*, a software designed to build optimal meshes, for creating the vertices, edges, and faces. While this method has led to more progress than the Reynwar Method did, the two obstacles mentioned in the introduction remain. The first, and more fundamental of the two, is in regards to creating meshes that result in *Surface Evolver* energies lower than analytic results predict. While I have created highly accurate meshes for 1 inclusion system, and believe similarly accurate ones can be created for the 2 inclusion system, this is heavily based on trial and error as opposed to clear qualities that lead to a mesh being accurate. A potential first step would be to play with this approach for the 1 inclusion system to become comfortable with how changes to the initial *Gmsh* file lead to changes in the *Surface Evolver* energy in the hopes of identifying clear qualities of a "good" mesh. The second obstacle is that of polar angle tilts above 5°. Currently, as the polar angle tilt increases, the faces between the border of the inclusion and the mirror plane of the mesh gradually become stretched. This should be surmountable with changes to the 'polar_change.cc' script used for each 0.05° (currently) tilt change. The mesh may also have to be refined to a higher level in *Gmsh* in this region, but moving the x, y, and z values of problem faces' vertices closer to the inclusion in 'polar_change.cc' would at least improve on the present method allowing for larger polar tilt angles, if not entirely solve the problem. If simply moving the vertices closer to the inclusion causes unsatisfactory stretches closer to the mirror plane, use of the *Surface Evolver* command *refine edges* on edges in the middle section between the inclusion and mirror plane may be useful. These are my current thoughts on progress here, however, and quite probably not sufficient to render this method complete.

2.3 Future

In each of the previous subsections I have mentioned the current obstacles of the method and hopefully provided a minor prod towards overcoming these obstacles. In the grand scheme of things, both methods would be wonderful to have functional for checking results against each other. That being said, I think that anyone continuing this project should first take a stab at the Reynwar Method, both because it requires getting familiar only with *Surface Evolver* and because I feel as if the current obstacle is relatively minor, though it dead-ended me. Perhaps a new perspective on my scripts for the Reynwar Method will quickly identify a small change that renders this method fully functional. In the case that a fix does not come so quickly, however, I feel that the Osman-Lucas Method has a less do-or-die predicament. The obstacle of limited polar angle tilting will require time and effort, but should yield improvements more predictably than a potentially unending dive into the black box that is *Surface Evolver* in pursuit of the Reynwar Method.

3 Specifics on use of Methods and *Surface Evolver*

3.1 *Surface Evolver*

3.1.1 Energy

The most fundamental part of using *Surface Evolver* to numerically calculate curvature-mediated interactions between membrane proteins depends on the energy present in the surface that is being minimized. By default, energy is solely based on surface area which can simply be removed by adding the line:

set facet tension 0

at the end of the file, after a line with the single word *read*. The energy that we would then like to use is called *star_sq_mean_curvature*, which can be used by inserting the following line (I do so at the top of the file):

quantity starsq energy method star_sq_mean_curvature

where *starsq* is a variable. Vertices can now have *starsq* set to them, making *Surface Evolver's* calculation of square mean curvature energy at this vertex contribute to the total energy, and making *hessian_seek* attempt to minimize this energy. Again, a look through the file 'FixedProtein14.txt' should provide a reference for these basics.

3.1.2 Viewing

Surface Evolver offers a decent way to view the current state of a mesh, which can be vital for determining issues with a method. Much of use of the viewer is covered quite well by documentation at `http://facstaff.susqu.edu/brakke/evolver/html/graphics.htm`. These are the few commands I used by far most:

z: zoom in or out, clicking and dragging mouse right or left respectively

t: translate the mesh by clicking and dragging

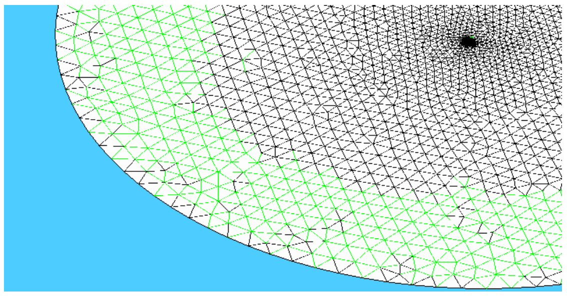
r: rotate the mesh about a focal point with click and drag

shift+f: after right clicking on a vertex (vertex number shows up in *Surface Evolver* command line), makes this vertex the new center about which rotation and zoom occurs

I would like to mention an additional technique that I found very useful: changing edge or face colors based on specific criteria. This can be used to identify where on the mesh edges with large dihedral angles exist, or even which edges have a vertex with particularly high energy (remember, the square mean curvature energy is calculated for vertices). A sample command for this is:

foreach edge ee DO{ IF ee.vertex[1].starsq > *energyamount* THEN set edge ff color green where ff.id = ee.id; IF ee.vertex[2].starsq > *energyamount* THEN set edge gg color green where gg.id = ee.id}

This command will set all edges green where either vertex has a starsq energy above *energyamount*, a variable to be replaced by a real number. This can be seen in the following image, where clearly nearly all of the outer vertices house energy higher than the *energyamount* specified in the command:



If energy amount is too low and many edges satisfy this requirement, the coloring of the edges may take so long that it crashes *Surface Evolver*, so I would generally start at about 10% of the total energy for *energyamount*, and repeat the command, each time reducing *energyamount* to about 10% of what it was for the previous command until some edges start to be colored. An alternative is to run the same command, replacing 'set edge ff color green' with 'list edge ff', and similar for gg. This will likely not crash *Surface Evolver*, and gives information on whether any edges would be colored, and if so how many.

3.1.3 Appending Commands

This was briefly mentioned above in section **3.1.1**, but a *Surface Evolver* file can be appended with commands to be enacted upon loading, or even to create new commands. To do so, after the file is otherwise complete, have a single line with *read*. After this line, each line is treated as a command to be entered, so having a line '*s*' and then a line '*q*' will open the viewing window and then return the command window to normal command mode from viewing command mode. To create new commands or adjust existing ones, create a line such as:

u :::= { 'u'; 'e'; set edge color green where id = 10 }

This will change the *u* command to first enact the original *u* command, and then enact the *e* command, and finally set the edge with id 10 to green (this will do nothing if it is already green). It is a nice way to make your life more efficient if you find yourself using the same commands in succession frequently.

3.2 Reynwar Method

Much of the Reynwar Method should be fairly straight forward within *Surface Evolver*, and hopefully looking at the text files in the 'Reynwar Method' can provide help for getting started. Beyond building a file to start with, as the supplemental material for Reynwar *et al.* describes, there are 4 main commands used:

hessian_seek: to do an energy minimization step
u: to equitriangulate all edges
refine edge: to refine all edges unless specified
delete edge: to remove all edges unless specified

In addition to these basic functions, there is a general format of using *refine edge* and *delete edge* that I found most able to capture the way Reynwar *et al.* describes their use – that is to target specific edges based on distance – to be:

foreach edge ee DO{ IF sqrt(($x0$ - (ee.vertex[1].x + ee.vertex[2].x)/2)^2 + ($y0$ - (ee.vertex[1].y + ee.vertex[2].y)/2)^2 + ($z0$ - (ee.vertex[1].z + ee.vertex[2].z)/2)^2) $<$ distance THEN refine edge where id = ee.id }

This is a sample format that will need to be changed to fit need, but the significant basics for looking over all the edges to check if they should be refined or deleted is 'foreach edge ee DO{ IF ... THEN ...}', where ee is simply a variable. In the case above, if $x0$, $y0$, and $z0$ are replaced with the coordinates for the center of the inclusion, then this line of code refines all edges whose midpoint is within *distnace* units of the center of the inclusion. The Reynwar Method heavily uses *refine edge* and *delete edge* in this manner based on distance from the center of the inclusion at first, and then refines based on the dihedral angle of each edge. Edges in *Surface Evolver* fortunately have a dihedral attribute that can be accessed with ee.dihedral (again, ee being a variable for an edge), which can then simply be used with the *refine edge* command as so: *refine edge ee where ee.dihedral* $> x$, where x is an angle in radians. The command *historgram(edge,dihedral)* yields a numerical histogram of dihedral angles and the number of edges that fit each range of dihedral angles.

3.3 Osman-Lucas Method

For both 1 inclusion and 2 inclusion models, this method relies almost entirely on running several python scripts, starting with a '.geo' file for *Gmsh*, and ending with a '.txt' file for *Surface Evolver*.

3.3.1 1 Inclusion

For 1 inclusion models, only the script 'runall.py' must be run, followed by minimizing energy of the output file in *Surface Evolver*. A sample run of this script is:

python runall.py 1_inclusionGmsh/6gen4ring10.geo 1_inclusion/newSEfile.txt full 0 10 0 10

This will take '6gen4ring10.geo', put it through *Gmsh*, put the file from *Gmsh* through 'vtk_poly.py' which reformats it, and finally put it through 'bc_flags.cc', which does necessary conversions to make 'newSEfile.txt' *Surface Evolver*-ready. The above command produces a full (fully circular, no mirror plane) mesh with an inclusion centered at $x = 0$, a contact angle of 10°, a polar tilt angle of 0°, and an outer radius of 10. The generalized syntax for the 'runall.py' command above is:

python runall.py [inputfile] [outputfile] [outer circle type (half or full)] [x value of inclusion center] [contact angle of inclusion] [polar angle of inclusion] [outer radius]

Be careful that the outer radius entered in this command matches the outer radius of the *Gmsh* file, and note that all output file names are prepended with 'fromGMSH/' by 'runall.py'. This output file is now ready to be opened in *Surface Evolver* and have its energy minimized via *hessian_seek*. Files from 'bc_flags.cc' (and so 'runall.py as well) are also appended with a command change that I found useful, that is the *E* command will run the equangulation command and print the energy twice, run *hessian_seek* three times, then run the equangulation command and print the energy twice again.

Starting from a *Gmsh* file, this is all that is necessary for the Osman-Lucas Method for a 1 inclusion model.

3.3.2 2 Inclusion

For 2 inclusion models, the first step is the same as for 1 inclusion models, that is running 'runall.py' with a *Gmsh* file, though the keyword *half* should be used instead of *full* to indicate it is to be a half-circle mesh, and the *Gmsh* file used must of course be set up for a 2 inclusion model. After using 'runall.py', however, 'tilt_convert.py' must be used to move the inclusion through various polar angle tilts and compare their energy. A sample run of these scripts for a 2 inclusion system is:

python runall.py 2_inclusionGmsh/halfring_2_1280.geo 2_inclusion/newSEfile.txt half 2 10 0 1280

python tilt_convert.py 2_inclusion/newSEfile.txt newSEfilemesh.txt newSEfileenergy.txt 2

The first command will take the file 'halfring_2_1280.geo', put it through 'runall.py' as described in **3.3.1** and yield a *Surface Evolver*-ready file with a half (semi-circular, with mirror plane) mesh with an inclusion centered at $x = 2$, a contact angle of $10°$, a polar tilt angle of $0°$, and an outer radius of 1280. The second command then takes that output file puts it through steps that increase the polar tilt angle, check the energy, prints both to 'newSEfileenergy.txt' (which is stored in the 'energy' folder), and repeats this process until getting to the final polar tilt angle, in this case $2°$. The polar tilt step size is currently $0.05°$, a fairly arbitrary choice. Brief testing with a smaller step size ($0.01°$) had a very minor effect on the minimum energy found. The generalized syntax for the 'runall.py' command is as described in **3.3.1** as well, and the generalized syntax for 'tilt_convert.py' is:

tilt_convert.py [inputfile] [outputfile] [energyfile] [polar angle change]

Note that all input file names are prepended with 'fromGMSH/' by the program, all output file names are prepended with 'mesh/', and all energy file names are prepended with 'energy/'. After completing the commands above, the file 'newSEfileenergy.txt' will hold the important data of the minimized energy of the surface at each polar angle tilt tested (between $0°$ and $2°$). Note these energies are for the semi-circular model, they must be doubled to acquire the energy for the entire 2 inclusion system. Only a single *Surface Evolver* file will result, the one corresponding to the final polar tilt angle, named 'newSEfilemesh.txt' above. To examine the surface corresponding to the minimum energy surface, 'tilt_convert.py' must be run again to the angle that was previously found to have the minimum energy. Additionally, when trying to open any resulting file from 'tilt_convert.py' in *Surface Evolver*, first open the mesh as a text file, and remove the last 4 lines, those reading:

```
E
E
dump
q
```

These lines are how 'tilt_convert.py' can repeatedly get the minimized energy for each tilt angle and move to the next, but will also cause the *Surface Evolver* to close just after opening. These lines being the source of minimized energy for each tilt angle also mean that if surprising behavior arises, this may be an area to check. Due to the minimal change in tilt angle for each iteration, the surface should not require many energy minimizing steps, but it is possible there are scenarios where these six (each E includes 3 *hessian_seek*) are not enough.

If the Osman-Lucas method for a 2 inclusion system is entirely functional, running the pair of commands at the beginning of this section, followed by finding the minimum value in the energy output file should reveal (half of) the minimum energy of such a system, and the polar tilt angle of the inclusions at which it occurs.

3.4 Folder Organization

Throughout this project, nomenclature practices have changed in response to changes in the project. I have placed a text file titled 'info.txt' in each folder that briefly describes the contents of the folder and the practice used for naming files.