Deriving Vesicle Shapes Using Spontaneous-Curvature Model

Rui Fang and Stephanie Zhang

December 2018

1 Introduction

All the traces of the first Earthlings have vanished. As a result, how life originated and evolved remains one of the grand challenges of science. Prior to the evolution of ever more complex enzymes, simple chemistry is thought to lead to the emergence of life and in particular, self-replicating membrane compartment [1]. It is critical to understand how protocell membranes can replicate and allow genetic polymer to be propagated down many generations.

As with modern cell membranes, protocell membranes are composed of amphiphiles that are arranged in a bilayer. However, the similarities do not extend to their growth and division. Although modern eukaryotic cell membranes are comprised mostly of phospholipids, primitive cell membranes were more likely composed of fatty acids. Fatty acids have a much higher likelihood of being prebiotically available, and have been found in carbonaceous chondrites, as well as in simulations of early Earth conditions with Fischer-Tropsch type synthesis. There are some key differences between fatty acid and phospholipid membranes. Compared to phospholipids of similar chain length, fatty acids move rapidly between the two bilayer leaflets, between vesicles, and being free in solution. Growing and dividing a fatty acid membrane without specialized proteins remains a challenge for synthetic biologists and origins of life researchers alike.

Given these differences, we try to build the theoretical understanding of the morphology and morphology transformations of phospholipid vesicles in this study. We hope that this can provide the tools for understanding morphological transitions in fatty acid systems.

2 Theories and Methods

Two variants of a continuum description have been introduced and studied: (i) the spontaneous-curvature model [1] and (ii) the bilayer-coupling model [2]. In particular, the bilayer-coupling model includes an additional constraint for the area difference of the two monolayers. Due to the time constraint, in this study,

we will focus on the spontaneous-curvature model, which has been introduced by Helfrich [2] and systematically studied by Seifert [3].

2.1 The Spontaneous-Curvature Model

According to [3], the bending energy of a vesicle with surface area A and volume V is given by

$$F_b \equiv \frac{\kappa}{2} \oint (C_1 + C_2 - C_0)\, dA + \kappa_G \oint C_1 C_2 dA, \tag{2.1}$$

where κ is the bending rigidity, κ_G is the Gaussian bending rigidity, C_1 and C_2 are the two principal curvatures, and C_0 is the spontaneous curvature accounting for a possible asymmetry of the bilayer membrane. The second term gives the integrated Gaussian curvature which is constant for topologically equivalent shapes. In this project, we will focus on shapes which share the same topology as a sphere and, thus, omit this term. Now the equation becomes

$$F_b = \frac{\kappa}{2} \oint (C_1 + C_2 - C_0)^2\, dA. \tag{2.2}$$

To understand the shape transformations of modern phospholipid vesicles, our preliminary goal is to determine the vesicle shape of the lowest bending energy for a given area A and volume V. This can be achieved via Lagrange multipliers, which finds the local maxima or minima of a function subject to equality constraints. Introducing the multipliers Σ and P, we obtain

$$\delta F \equiv \delta \left(F_b + \Sigma A + PV\right) = 0, \tag{2.3}$$

where δ denotes variation with respect to the shape.

To simplify the search, we restrict the solutions to axisymmetric shapes. We parameterize this shape with coordinates (S, ϕ) (see Figure 1). Here S denotes the arclength along the contour measured from the north pole of the shape, ϕ is the azimuthal angle. The shape is described by the tilt angle $\psi(S, \phi)$. Since the shape is axisymmetric by assumption, the dependence on ϕ can be removed. We also use the coordinates X and Z which are perpendicular and parallel to the axis of symmetry respectively. The shape can also be parameterized with $Z(X)$.

With this system of parameterization, there exist the following geometrical relations:

$$\dot{X} \equiv \frac{dX}{dS} = \cos\psi \tag{2.4a}$$

$$\dot{Z} \equiv \frac{dZ}{dS} = -\sin\psi \tag{2.4b}$$

$$C_1 = \dot{\psi} \equiv \frac{d\psi}{dS} \tag{2.4c}$$

$$C_2 = \frac{\sin\psi}{X} \tag{2.4d}$$

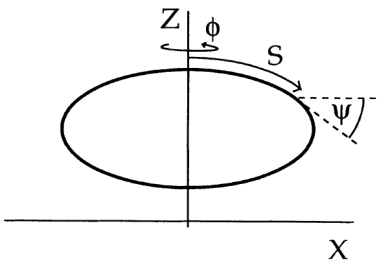


Figure 1: Parametrization of vesicle shape [3].

The total energy F regarding the above parameterization can be expressed as

$$F = 2\pi\kappa \int_0^{S_1} L\left(\psi, \dot{\psi}, X, \dot{X}, \gamma\right) dS, \tag{2.5}$$

with the Lagrange function

$$L \equiv \frac{X}{2}\left[\dot{\psi} + \frac{\sin\psi}{X} - C_0\right]^2 + \overline{\Sigma}X + \frac{\overline{P}}{2}X^2\sin\psi + \gamma\left(\dot{X} - \cos\psi\right), \tag{2.6}$$

and rescaled parameters

$$\overline{\Sigma} \equiv \frac{\Sigma}{\kappa}, \tag{2.7}$$

$$\overline{P} \equiv \frac{P}{\kappa}. \tag{2.8}$$

The term $\gamma = \gamma(S)$ is the Lagrange parameter function which enforces the equality between $\dot{X}$ and $\cos\psi$.

2.2 Shape Equations and Boundary Conditions

Taken together, the Euler-Lagrange equations for the F give rise to the set of shape equations, which was found using `euler_equations` of the sympy module in python:

$$\dot{\psi} = U, \tag{2.9a}$$

$$\dot{U} = -\frac{U}{X}\cos\psi + \frac{\cos\psi\sin\psi}{X^2} + \frac{\gamma}{X}\sin\psi + \frac{\overline{P}X}{2}\cos\psi, \tag{2.9b}$$

$$\dot{\gamma} = \frac{(U - C_0)^2}{2} - \frac{\sin^2\psi}{2X^2} + \overline{P}X\sin\psi + \overline{\Sigma}, \tag{2.9c}$$

$$\dot{X} = \cos\psi. \tag{2.9d}$$

The boundary conditions are

$$\psi(0) = 0, \tag{2.10a}$$
$$\psi(S_1) = \pi, \tag{2.10b}$$
$$X(0) = 0, \tag{2.10c}$$
$$X(S_1) = 0, \tag{2.10d}$$
$$\gamma(0) = 0, \tag{2.10e}$$
$$\gamma(S_1) = 0. \tag{2.10f}$$

2.3 Solve the Shape Equations Numerically

The shape equations (2.9) are a set of nonlinear ordinary differential equations (ODEs). To determine the stationary shapes, we must solve the boundary value problem associated with the ODEs subject to the boundary conditions (2.10).

For fixed $\overline{P}, \overline{\Sigma}, C_0$, we can obtain a complete set of solutions with the following procedure:

1. Integrate the equations (2.9) with initial values $\psi(0) = 0$, $X(0) = 0$, $\gamma(0) = 0$, and for a wide range of $U(0)$'s. For each $U(0)$, terminate the integration if $\psi(S) > 3\pi$, or $\psi(S) < -2\pi$, or S reaches an arbitrary maximal value $S_{\max}$.

2. For each tentative solution corresponding to a $U(0)$, detect where and for how many times that $\psi = \pi$. Define $S_1^{(n)}$ by $\psi(S_1^{(n)}) = \pi$ with $n = 1, 2, 3, ...$

3. Plot $X(S_1^{(n)})$ versus $U(0)$. Estimate for which $U(0)$'s that $X(S_1^{(n)}) = 0$. These are solutions that satisfy the boundary conditions (2.10).

4. The estimation in the above step is rough. For each one of the possible correct solutions, to get the exact value of $U(0)$, sweep through several values of $U(0)$ in a small range around the estimated value. Plot the solutions. It is often obvious to identify two classes of behaviors in the solutions. Utilize bracketing method (binary search) to obtain the correct solution.

Because $X(0) = 0$ appears in the denominator of several terms in (2.9), in practice the numerical solver is not able to integrate from $S = 0$. We thus begin the integration from a very small $S = S_0$ and approximate the initial values of ψ, U, X and γ by taking the Taylor expansion around $S = 0$.

Let

$$\psi(S) = \sum_{n=1}^{\infty} a_n S^n, \tag{2.11a}$$

$$X(S) = \sum_{n=1}^{\infty} b_n S^n, \tag{2.11b}$$

$$\gamma(S) = \sum_{n=1}^{\infty} c_n S^n. \tag{2.11c}$$

Then

$$U(S) = \dot{\psi}(S) = a_1 + \sum_{n=2}^{\infty} n a_n S^{n-1}. \tag{2.12}$$

Substituting these into the shape equations (2.9) and equating the coefficients, we obtain

$$\psi(S) = a_1 S + \mathcal{O}\left(S^3\right), \tag{2.13a}$$

$$U(S) = a_1 + \mathcal{O}\left(S^2\right), \tag{2.13b}$$

$$\gamma(S) = \left(\frac{C_0^2}{2} - C_0 a_1 + \overline{\Sigma}\right) S + \mathcal{O}\left(S^3\right), \tag{2.13c}$$

$$X(S) = S + \mathcal{O}\left(S^3\right). \tag{2.13d}$$

Eliminate higher order terms. The initial values are then approximated by

$$\psi(S = S_0) = a_1 S_0, \tag{2.14a}$$

$$U(S = S_0) = a_1, \tag{2.14b}$$

$$\gamma(S = S_0) = \left(\frac{C_0^2}{2} - C_0 a_1 + \overline{\Sigma}\right) S_0, \tag{2.14c}$$

$$X(S = S_0) = S_0. \tag{2.14d}$$

Example

To give an example, we here present the procedure of finding one stationary shape solution for fixed constants $\overline{P} = 1$, $\overline{\Sigma} = -1.1$, $C_0 = 0$.

Figure 2 show the plot of $X(S_1^{(n)})$ versus $U(0)$. We can see that solutions occur at several places, namely, when $U(0)$ is close to $-0.55, 0.1, 0.45, 0.6, 0.95$.

Focusing on the first one, we sweep through different $U(0)$ values around $U(0) = -0.55$ and plot the solutions in Figure 3. We notice that ψ goes upwards for $U(0) \leq -0.558$ and goes downwards for $U(0) \geq -0.556$. Hence we perform a binary search between $U(0)_1 = -0.558$ and $U(0)_2 = -0.556$, using a criteria that if $\psi(S = 10, U(0)) > \pi$, then let $U(0) = U(0)_1$ otherwise let $U(0) = U(0)_2$.

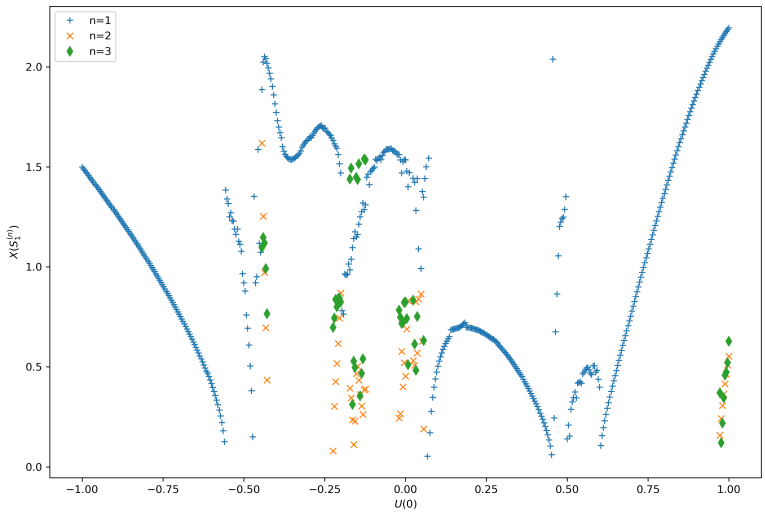


Figure 2: Plot of $X(S_1^{(n)})$ versus $U(0)$.

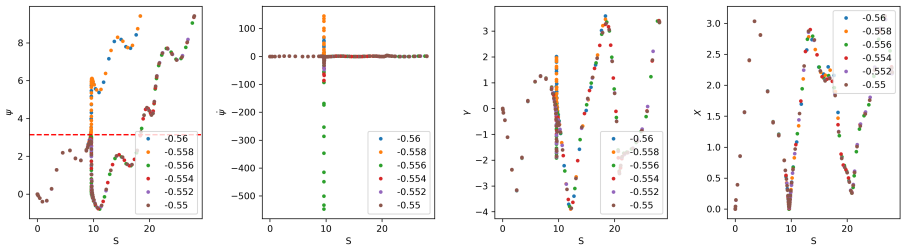


Figure 3: Sweeping through different $U(0)$ values around $U(0) = -0.55$.

This yields the solution as shown in Figure 4. The correct value of $U(0)$ is -0.556399201750464. From the solution we can reconstruct the shape contour using $Z(S) = \int_0^S (-\sin\psi(S'))\, dS'$ and $X(S)$. The corresponding shape is shown in Figure 5.

3 Results

We derived solutions for three sets of constants:

- $\overline{P} = 1, \overline{\Sigma} = -1.1, C_0 = 0$. Solutions are plotted in Figure 6.
- $\overline{P} = 1, \overline{\Sigma} = -0.5, C_0 = 0$. Solutions are plotted in Figure 7.

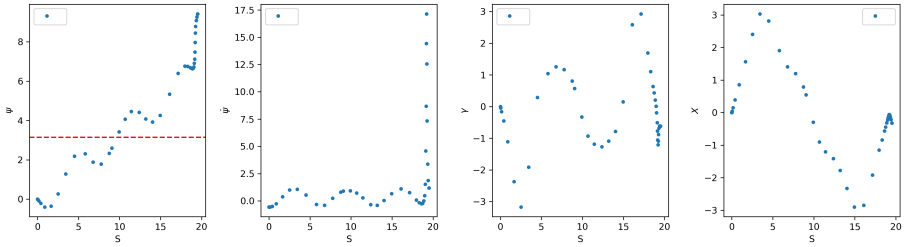


Figure 4: The solution.

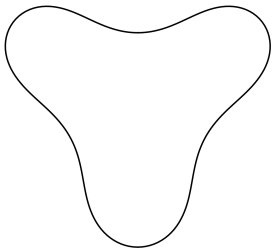


Figure 5: The corresponding shape.

- $\overline{P} = 1, \overline{\Sigma} = 1.0, C_0 = 0$. Solutions are plotted in Figure 8.

These covered many representative shapes including sphere, prolate sphere, oblate sphere, stomatocyte, inverted sphere and self-intersecting discocyte.

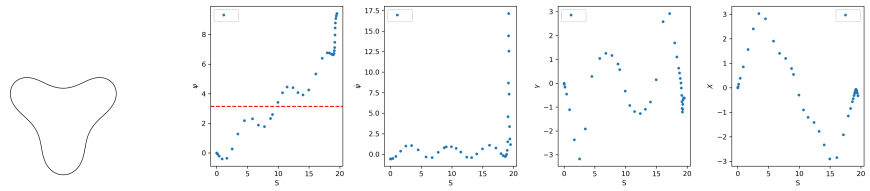
Compared to [3], our results successfully matched the ones in Figure 16 and Figure 2(a). Due to time constraint, we only generated the results for a limited set of constants. In principal, we can employ this procedure repeatedly to derive a complete phase diagram. This is left as a future study.

4 Conclusion and Future Directions

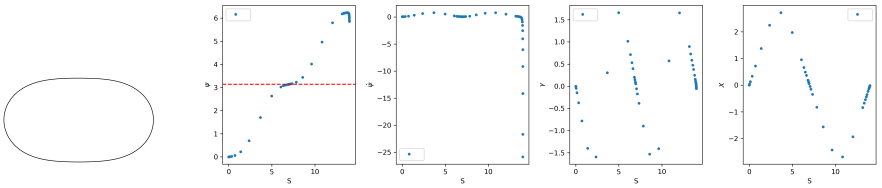
In conclusion, we have provided a simple study of phospholipid vesicle shapes and their transformations. This provides the framework for understanding the vesicle morphology, and potentially protocell studies.

To employ this work for the fatty acid systems, further action is required: adjusting the equations to describe the flip-flop behaviors, which will expand the routes for model protocell division. In addition, we could build such system to probe how the environmental triggers transform the shape of the vesicles.

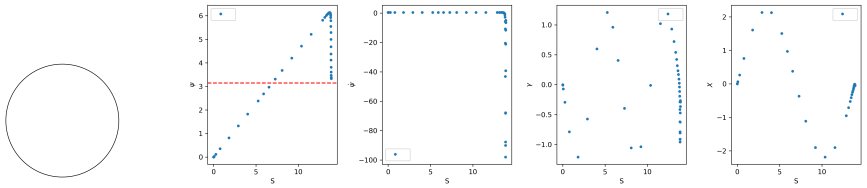
Our search for life on other planets is shaped by what we know of life on Earth. Understanding the shape transformations of the membrane compart-



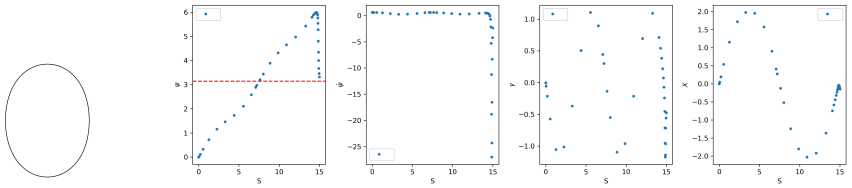
(a) $U(0) = -0.556399201750464$



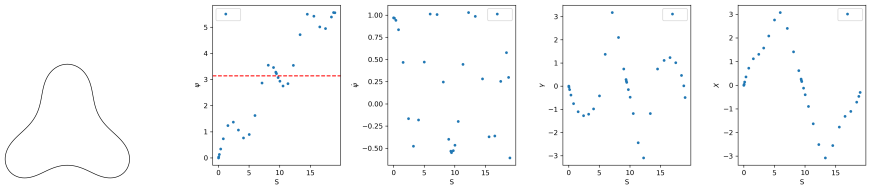
(b) $U(0) = 0.067864763964811$



(c) $U(0) = 0.4541615196686325$



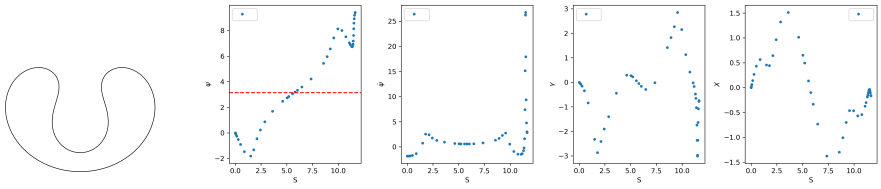
(d) $U(0) = 0.6001853059954738$



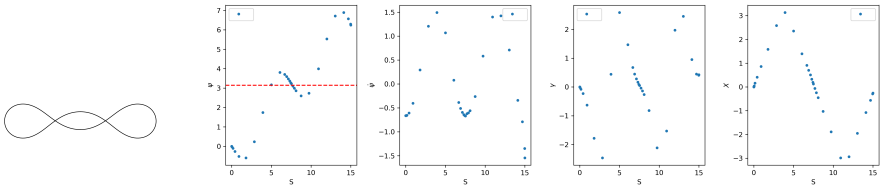
(e) $U(0) = 0.9688285481506609$

Figure 6: Solutions for $\overline{P} = 1, \overline{\Sigma} = -1.1, C_0 = 0$.

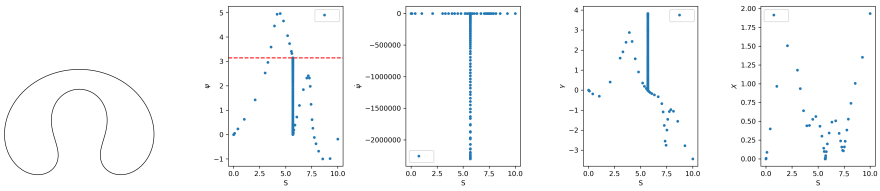
ments for the origin of life, as we know it, can inform us how the earliest life may have propagated over many generations.



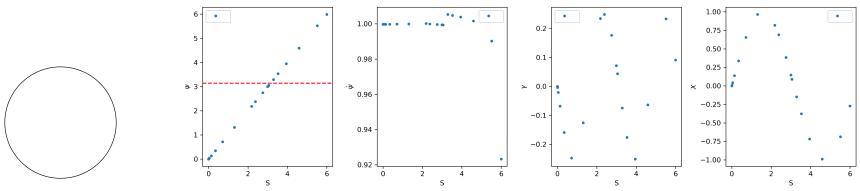
(a) $U(0) = -1.8477405209981224$



(b) $U(0) = -0.661554195917779$

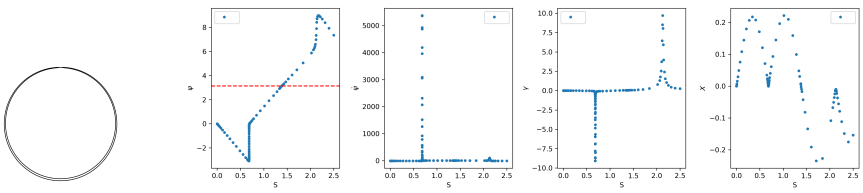


(c) $U(0) = 0.5802963392822205$



(d) $U(0) = 0.9997786947310489$

Figure 7: Solutions for $\overline{P} = 1, \overline{\Sigma} = -0.5, C_0 = 0$.



(a) $U(0) = -4.593793448929572$

Figure 8: Solutions for $\overline{P} = 1, \overline{\Sigma} = 0.1, C_0 = 0$.

References

[1] Jack W Szostak. The narrow road to the deep past: In search of the chemistry of the origin of life. *Angewandte Chemie International Edition*, 56(37):11037–11043, 2017.

[2] Wolfgang Helfrich. Elastic properties of lipid bilayers: theory and possible experiments. *Zeitschrift für Naturforschung C*, 28(11-12):693–703, 1973.

[3] Udo Seifert, Karin Berndl, and Reinhard Lipowsky. Shape transformations of vesicles: Phase diagram for spontaneous-curvature and bilayer-coupling models. *Physical Review A*, 44(2):1182, 1991.

Appendix: Python code for the project

```
#Set up the environment
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from JSAnimation import IPython_display
from sympy import *
from scipy.integrate import solve_bvp, solve_ivp, quad

#Define symbols and functions
S = Symbol(r'S')
psi = Function(r'\psi')
X = Function(r'X')
gamma = Function(r'\gamma')
C0, SigmaBar, PBar = symbols(r'C_0 \overline{\Sigma} \overline{P}')

#Define the Lagrangian
L = X(S)/2*(psi(S).diff(S)+sin(psi(S))/X(S)-C0)**2 + SigmaBar*X(S)
    + PBar/2*X(S)**2*sin(psi(S)) + gamma(S)*(X(S).diff(S)-cos(psi(S
    )))

#Use euler_equations to find the Euler-Lagrange equations
LEeqns = euler_equations(L, [psi(S), X(S), gamma(S)], S)
LEeqns
```

Listing 1: Derivation of the shape equations

```
S, psi, X, gamma = symbols(r'S \psi X \gamma')
C0, SigmaBar, PBar = symbols(r'C_0 \overline{\Sigma} \overline{P}')

n_terms = 3
a = [symbols('a'+str(i)) for i in range(1, n_terms+1)]
b = [symbols('b'+str(i)) for i in range(1, n_terms+1)]
c = [symbols('c'+str(i)) for i in range(1, n_terms+1)]

psi = sum(a[i]*S**(i+1) for i in range(n_terms))
X = sum(b[i]*S**(i+1) for i in range(n_terms))
gamma = sum(c[i]*S**(i+1) for i in range(n_terms))

```

```
shape_eqns = [
    psi.diff(S).diff(S) + psi.diff(S)/X*cos(psi) - cos(psi)*sin(psi)/X**2 - gamma/X*sin(psi) - PBar*X*cos(psi)/2,
    gamma.diff(S) - (psi.diff(S)-C0)**2/2 + sin(psi)**2/(2*X**2) - PBar*X*sin(psi) - SigmaBar,
    X.diff(S) - cos(psi)
]

eq1_order0 = shape_eqns[0].series(S, 0, 1).removeO()
eq1_order1 = shape_eqns[0].series(S, 0, 2).removeO() - eq1_order0
eq2_order0 = shape_eqns[1].series(S, 0, 1).removeO()
eq2_order1 = shape_eqns[1].series(S, 0, 2).removeO() - eq2_order0
eq3_order0 = shape_eqns[2].series(S, 0, 1).removeO()
eq3_order1 = shape_eqns[2].series(S, 0, 2).removeO() - eq3_order0

solve([eq1_order0, eq1_order1, eq2_order0, eq2_order1, eq3_order0, eq3_order1], set(a+b+c))
```

Listing 2: Taylor expansion

```
def shape_func(S, y, Sigmabar, Pbar, C0):
    '''System of ODEs'''

    # Variables. psi and dpsi are viewed as two variables.
    (psi, dpsi, gamma, X) = y

    ddpsi = -dpsi*np.cos(psi)/X +
    np.cos(psi)*np.sin(psi)/(X**2)
    + gamma*np.sin(psi)/X +  Pbar*X*np.cos(psi)/2
    dgamma = (dpsi-C0)**2/2 - np.sin(psi)**2/(2*X**2) + Pbar*X*np.sin(psi)
    + Sigmabar
    dX = np.cos(psi)

    # Return derivatives of every variable
    dy = (dpsi, ddpsi, dgamma, dX)

    return dy


def y0(S0, U0, Sigmabar, C0):
    '''Initial conditions'''

    return [U0*S0, U0, (C0**2/2-C0*U0+Sigmabar)*S0, S0]


def psi_reach_pi(S, y):
    '''Event function for detecting when psi reaches pi'''

    return y[0]-np.pi


def psi_reach_3pi(S, y):
    '''Event function for detecting when psi reaches 3pi'''

    return y[0]-3*np.pi
psi_reach_3pi.terminal = True

```

```
38
39 def psi_reach_minus_2pi(S, y):
40     '''Event function for detecting when psi reaches -2pi'''
41
42     return y[0]+2*np.pi
43 psi_reach_minus_2pi.terminal = True
44
45
46 def plot_solution(sols, labels):
47     ylabels = [r'$\psi$', r'$\dot{\psi}$', r'$\gamma$', r'$X$']
48
49     fig, axes = plt.subplots(1, 4, figsize=(14, 4))
50     for i in range(4):
51         for sol, label in zip(sols, labels):
52             axes[i].plot(sol.t, sol.y[i], '.')
53         if i == 0:
54             axes[i].axhline(y=np.pi, color='r', linestyle='--')
55         axes[i].set_xlabel('S')
56         axes[i].set_ylabel(ylabels[i])
57         axes[i].legend(labels)
58     plt.tight_layout()
59     plt.show()
60
61
62 def generate_contour(sol):
63     psi_of_S = lambda S: sol.sol(S)[0]
64     X_of_S = lambda S: sol.sol(S)[3]
65     Z_of_S = lambda S: quad(lambda s: -np.sin(psi_of_S(s)), 0, S)
       [0]
66
67     if len(sol.t_events[0]) > 0:
68         ss = np.linspace(0, sol.t_events[0][-1], 100)
69     else:
70         ss = np.linspace(0, sol.t[-1], 100)
71
72     xx = np.array([X_of_S(S) for S in ss])
73     zz = np.array([Z_of_S(S) for S in ss])
74
75     return [xx, zz]
76
77 def plot_contour(coords):
78     x, z = coords
79     plt.plot(x, z, 'k-')
80     plt.plot(-x, z, 'k-')
81     plt.axis('equal')
82     plt.axis('off')
```

Listing 3: Functions for solving and plotting solutions

```
1 Sb = -1.1
2 Pb = 1.
3 C0 = 0.
4
5 S0 = 1e-7
6 S1 = 100
7 U0_list = np.linspace(-1, 1, 501)
8
9 sols = []
```

```
for U0 in U0_list:
    sol = solve_ivp(lambda S, y: shape_func(S, y, Sb, Pb, C0),
    (S0, S1), y0(S0, U0, Sb, C0), method='RK45',
    dense_output=True, events=[psi_reach_pi,
    psi_reach_3pi, psi_reach_minus_2pi])
    sols.append(sol)

U0_n_geq1 = np.array([U0 for sol, U0 in zip(sols, U0_list)
if len(sol.t_events[0]) > 0])
U0_n_geq2 = np.array([U0 for sol, U0 in zip(sols, U0_list)
if len(sol.t_events[0]) > 1])
U0_n_geq3 = np.array([U0 for sol, U0 in zip(sols, U0_list)
if len(sol.t_events[0]) > 2])

S1_n_geq1 = np.array([sol.t_events[0] for sol in sols
if len(sol.t_events[0]) > 0])
S1_n_geq2 = np.array([sol.t_events[0] for sol in sols
if len(sol.t_events[0]) > 1])
S1_n_geq3 = np.array([sol.t_events[0] for sol in sols
if len(sol.t_events[0]) > 2])

y_n_geq1 = np.array([sol.sol(sol.t_events[0])[:, 0]
for sol in sols if len(sol.t_events[0]) > 0])
y_n_geq2 = np.array([sol.sol(sol.t_events[0])[:, 1]
for sol in sols if len(sol.t_events[0]) > 1])
y_n_geq3 = np.array([sol.sol(sol.t_events[0])[:, 2]
for sol in sols if len(sol.t_events[0]) > 2])

print(U0_n_geq1.shape, S1_n_geq1.shape, y_n_geq1.shape)
print(U0_n_geq2.shape, S1_n_geq2.shape, y_n_geq2.shape)
print(U0_n_geq3.shape, S1_n_geq3.shape, y_n_geq3.shape)

plt.figure(figsize=(12, 8))
plt.plot(U0_n_geq1, y_n_geq1[:, 3], '+-', label='n=1')
plt.plot(U0_n_geq2, y_n_geq2[:, 3], 'x', label='n=2')
plt.plot(U0_n_geq3, y_n_geq3[:, 3], 'd', label='n=3')
plt.xlabel(r'$U(0)$')
plt.ylabel(r'$X(S_1^{(n)})$')
plt.legend()
plt.show()
```

Listing 4: Code for creating Figure 2

```
Sb = -1.1
Pb = 1.
C0 = 0.

S0 = 1e-7
S1 = 40.
U0_list = np.linspace(-0.56, -0.55, 6)

sols = []
for U0 in U0_list:
    sol = solve_ivp(lambda S, y: shape_func(S, y, Sb, Pb, C0), (S0,
     S1), y0(S0, U0, Sb, C0), method='RK45', dense_output=True,
    events=[psi_reach_pi, psi_reach_3pi, psi_reach_minus_2pi])
    print(U0, sol.t_events)
    sols.append(sol)
```

```
14
15 plot_solution(sols, U0_list)
```

Listing 5: Code for creating Figure 3

```
1  Sb = -1.1
2  Pb = 1.
3  C0 = 0.
4
5  S0 = 1e-7
6  S1 = 10.
7
8  U0_1 = -0.558
9  U0_2 = -0.556
10
11 while U0_2 - U0_1 > 1e-13:
12
13     U0 = (U0_1 + U0_2)/2
14
15     sol = solve_ivp(lambda S, y: shape_func(S, y, Sb, Pb, C0),
16                     (S0, S1),y0(S0, U0, Sb, C0),
17                     method='RK45',
18                     dense_output=True,
19                     events=[psi_reach_pi, psi_reach_3pi,
       psi_reach_minus_2pi])
20
21     if len(sol.t_events[0]) > 0:
22         U0_1 = U0
23     else:
24         U0_2 = U0
25
26 print(U0)
```

Listing 6: Code for performing the binary search

```
1  Sb = -1.1
2  Pb = 1.
3  C0 = 0.
4
5  S0 = 1e-7
6  S1 = 40.
7  U0 = -0.556399201750464
8
9  sol = solve_ivp(lambda S, y: shape_func(S, y, Sb, Pb, C0),
10                 (S0, S1), y0(S0, U0, Sb, C0),
11                 method='RK45',
12                 dense_output=True,
13                 events=[psi_reach_pi, psi_reach_3pi,
       psi_reach_minus_2pi])
14
15 plot_solution([sol], [None])
16
17 plot_contour(generate_contour(sol))
```

Listing 7: Code for creating Figure 4 and Figure 5