THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF PHYSICS

CHAOS IN SEMI-CLASSICAL COSMOLOGICAL MODELS

MARIA LAN BRESSAN
SPRING 2023

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Physics
with honors in Physics

Reviewed and approved* by the following:

Martin Bojowald
Professor of Physics
Thesis Supervisor

Richard Robinett
Professor of Physics
Honors Adviser

* Electronic approvals are on file.

Abstract

Chaotic behavior has been observed in cosmological models but hasn't been fully explored yet. The purpose of the project is to investigate chaotic behavior in various cosmological models. Adding quantum corrections to the cosmological models may affect the chaotic behavior. This will be done by numerically solving the equations of motion given by different cosmological models that exhibit chaos and observing the resulting trajectories. The goal is to better understand the chaotic behavior of cosmological models and how adjusting the coefficient to our quantum corrections affect the resulting behavior. In doing so, we can better understand parameters for the initial conditions of the universe and the fluctuations in the CMB radiation.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Dr. Martin Bojowald for being an excellent thesis advisor and for inspiring my interest in cosmology through his course on special and general relativity. His insightful comments, constructive feedback, and support have been instrumental in the success of this thesis.

I would also like to thank Dr. Richard Robinett for being the most wonderful academic advisor I could ask for. Furthermore, I would like to express my gratitude to the faculty, staff and students at Penn State's Department of Physics for making my undergraduate years extremely enriching both academically and socially.

Lastly, I wish to thank my parents for their efforts in instilling a passion for learning from an early age. Their sacrifices and encouragement have been critical in allowing me to pursue my academic goals and have provided me with a strong foundation for success.

Introduction

Cosmological models have been the subject of intense study for many decades, as they provide valuable insights into the early universe and its evolution. One particularly interesting phenomenon that has been observed in these models is chaotic behavior. While this behavior has been seen in various models, its implications and underlying mechanisms have not been fully explored. The purpose of this project is to investigate chaotic behavior in different cosmological models and assess how adding quantum corrections may affect this behavior. To achieve this, we will numerically solve the equations of motion for various early-universe models that exhibit chaotic behavior and analyze the resulting trajectories. The ultimate goal of this research is to gain a deeper understanding of the chaotic behavior of cosmological models, particularly with respect to the initial conditions of the universe and the fluctuations in the cosmic microwave background radiation. This paper begins with a general overview of main concepts in cosmology and then focuses on deriving the specific potential that we are investigating. We approximate this potential, adding a cubic term which is the focus of analysis. Through this, we investigate the impact of the coefficient of the cubic term in our approximation to chaotic behavior of this particular cosmological model, providing insights into the nature of the early universe and its evolution.

1.1 Geometry at Large Scales

At large scales, the universe is isotropic and homogeneous. Maps of the cosmic background radiation and the distribution of galaxies show that no one direction or position is preferred. There-

fore, the geometry of spacetime can be approximated by a line element of the form

$$ds^2 = -dt^2 + a^2(t)d\mathcal{L}^2 \tag{1.1}$$

where $d\mathcal{L}^2$ is the line element of a three-dimensional space that is time-independent, isotropic and homogeneous [2]. These are called Robertson-Walker metrics. There are three possibilities for Robertson-Walker metrics, the simplest of them being the flat Robertson-Walker metric. In polar coordinates, it is

$$ds^2 = -dt^2 + a(t)(dr^2 + r^2(d\theta^2 + \sin^2\theta d\phi^2). \tag{1.2}$$

The case where $a(t)$ increases with time describes an expanding universe where the physical distance between two objects is defined as

$$d(t) = a(t)d_{\text{coord}}. \tag{1.3}$$

We can then measure the Hubble constant

$$H_0 = \frac{\dot{a}(t_0)}{a(t_0)} \approx 72 \text{ km/s/Mpc} \tag{1.4}$$

from experimental values and approximate the age of the universe with the Hubble time, $t_H = 1/H_0 \approx 13.6$ Gyr.

1.2 Cosmological Fluids

In the Friedmann-Robertson-Walker (FRW) models, space is filled with three types of non-interacting cosmological fluids: pressureless matter, radiation, and vacuum. We can use this to determine how the scale factor $a(t)$ depends on time. Applying the first law of thermodynamics gives

$$\frac{d[\rho(t)a^3(t)]}{dt} = -p(t)\frac{d[a^3(t)]}{dt}. \tag{1.5}$$

A special case of the Friedman equation,

$$\dot{a}^2 - \frac{8\pi\rho}{3}a^2 = 0 \tag{1.6}$$

gives the relation between ρ, the total matter energy density, and a, the scale factor. In a matter dominated universe, $a(t) = (t/t_0)^{2/3}$, in a radiation dominated universe, $a(t) = (t/t_0)^{1/2}$, and in a vacuum dominated universe, $a(t) = e^{H(t-t_0)}$ [3].

Semi-Classical Model

2.1 Potential

In this chapter, we motivate the potential of a semi-classical cosmological model used in the analysis. We use a spatially isotropic cosmological model derived in Ref [1] with positive spatial curvature and an energy density given by

$$\rho(a) = \Lambda + \frac{\sigma}{a} + \rho_\phi \tag{2.7}$$

where $\Lambda < 0$ and $\sigma > 0$. Using this energy density, the Friedmann equation becomes

$$\frac{\dot{a}^2}{a^2} + \frac{k}{a^2} = \frac{8\pi F}{3}\left(\Lambda + \frac{\sigma}{a} + \rho_\phi\right) \tag{2.8}$$

and can be rewritten as

$$0 = \dot{a}^2 + \omega^2(a - \gamma/\omega)^2 + k - \gamma^2 - \frac{\tilde{p}^2}{a^2} = \dot{a}^2 + U_{\text{harmonic}}(a) - \frac{\tilde{p}^2}{a^4} \tag{2.9}$$

where

$$U_{\text{harmonic}}(a) = \omega^2(a - \gamma/\omega)^2 + k - \gamma^2 \tag{2.10}$$

and

$$\tilde{p} = \sqrt{\frac{4\pi G}{3}} p_\phi. \tag{2.11}$$

$U_{\text{harmonic}}(a)$ is a standard harmonic-oscillator potential with

$$\omega = \sqrt{-\frac{8\pi G\Lambda}{3}} \tag{2.12}$$

and

$$\gamma = \sqrt{-\frac{2\pi G\sigma^2}{3\Lambda}}. \tag{2.13}$$

We use canonical methods and therefore use the canonical momentum of a given by

$$p_a = -\frac{3}{4\pi G} a\dot{a}. \tag{2.14}$$

to substitute for the scale factor. We perform a canonical transformation from a, p_a to (α, p_α) using a logarithmic scale factor

$$\alpha = \ln(\omega\gamma a) \tag{2.15}$$

which also gives

$$p_\alpha = a p_a = -\frac{3}{4\pi G} a^2\dot{a}. \tag{2.16}$$

We then have the energy equation for (α, p_α) as

$$0 = \frac{16}{9}\pi^2 G^2 p_\alpha^2 + \frac{1}{\omega^4\gamma^4} e^{4\alpha} U_{\text{harmonic}}(a(\alpha)) - \tilde{p}^2. \tag{2.17}$$

The dynamical equation

$$0 = p_\alpha^2 + U_p(\alpha) \tag{2.18}$$

is then obtained with the potential

$$U_p(\alpha) = \frac{e^4\alpha}{\beta^2}\left(k - 2e^\alpha + \frac{e^{2\alpha}}{\gamma^2}\right) - p^2 \tag{2.19}$$

where β and p are defined as

$$\beta = \frac{4\pi G}{3}\omega^2\gamma^2 \tag{2.20}$$

and

$$p = \frac{3}{4\pi G}\tilde{p}. \tag{2.21}$$

2.2 Semi-Classical Dynamics

Semi-classical dynamics are obtained by interpreting variables like α and p_α as expectation values of corresponding operators in an evolving quantum state. If a potential is not harmonic, it means that its variables are coupled to fluctuations, correlations, and higher moments, which leads to dynamics in a configuration space of higher dimension. The coupling terms can be obtained by applying a Poisson bracket to the moments and using them in the Hamiltonian expectation value, calculated using the same state in which the moments were determined. We formulate the semi-classical description using the expectation values of basic operators in a state coupled to higher moments and fluctuations,

$$\Delta(\alpha^a p_\alpha^b) = \left\langle (\hat{\alpha} - \langle\hat{\alpha}\rangle)^a (\hat{p}_\alpha - \langle\hat{p}_\alpha\rangle)^b \right\rangle_{\text{symm}} \tag{2.22}$$

in symmetric ordering. We define the Poisson bracket to be

$$\left\{ \left\langle \hat{A} \right\rangle, \left\langle \hat{B} \right\rangle \right\} = \frac{\left\langle \left[\hat{A}, \hat{B}\right] \right\rangle}{i\hbar} \tag{2.23}$$

and extend it to moments using the Leibniz rule to obtain a phase-space structure. The Poisson bracket of moments is non-canonical meaning the Jacobi identity is not satisfied. For example, $\{\Delta(\alpha^2), \Delta(p_\alpha^2)\} = 4\Delta(\alpha p_\alpha)$. We apply a transformation from three dimensional space of second-order moments to new variables (s, p_s, U), such that

$$\Delta(\alpha^2) = s^2 \tag{2.24}$$

$$\Delta(\alpha p_\alpha) = s p_s \tag{2.25}$$

and

$$\Delta(p_\alpha^2) = p_s^2 + \frac{U}{s^2}. \tag{2.26}$$

Taylor expanding Eqn. (2.19) gives the momentum-corrected constraint

$$0 = \left\langle \hat{p_\alpha}^2 \right\rangle + \Delta(p_\alpha^2) + U_p(\langle\hat{\alpha}\rangle) + \Sigma_{n=2}^{\inf} \frac{1}{n!} \frac{d^n U_p(\langle\hat{\alpha}\rangle)}{d\,\langle\hat{\alpha}\rangle^n} \Delta(\alpha^n). \tag{2.27}$$

Including moments of second order and using the new variables (s, p_s, U), we have or semiclassical constraint,

$$0 = p_\alpha^2 + p_s^2 + \frac{U}{s^2} + U_p(\alpha) + \frac{1}{2} U_p^{''}(\alpha) s^2. \tag{2.28}$$

This can be approximated using an all-orders closure proposed in Ref [4] where

$$\Delta(\alpha^n) = s^n \tag{2.29}$$

for even n and

$$\Delta(\alpha^n) = 0 \tag{2.30}$$

for odd n. Using this closure, Eqn. (2.19) becomes

$$0 = p_\alpha^2 + p_s^2 + \frac{U}{s^2} + \frac{1}{2}\left(U_p(\alpha + s) + U_p(\alpha - s)\right). \tag{2.31}$$

The approximation is further improved when we include a cubic and quartic term in s such that

$$0 = p_\alpha^2 + p_s^2 + \frac{U}{s^2} + \frac{1}{2}\left(U_p(\alpha + s) + U_p(\alpha - s)\right) + \frac{1}{12} U^{''''} s^4 + c U^{'''} s^3. \tag{2.32}$$

The quartic term brings the approximation closer to gaussian form, such that $\Delta(\alpha^4) = 3s^4$ instead of $\Delta(\alpha^4) = s^4$, while the cubic term is the first non-gaussian contribution.

2.3 Cubic Term Coefficient

In Ref [1], the constraint in Eqn. (2.32) is studied without the final cubic term. The main focus of this analysis is to study how the addition of the cubic term and the adjustment of its coefficient affects the tunneling dynamics. Figure 2.1 shows the logarithm of the potential without the cubic

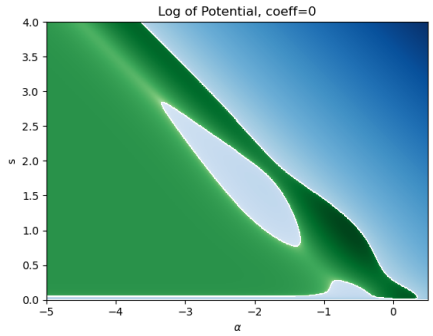

Figure 2.1: Log of Potential, c = 0.0

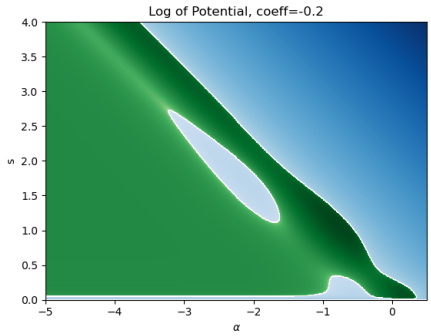

Figure 2.2: Log of Potential, c = -0.2

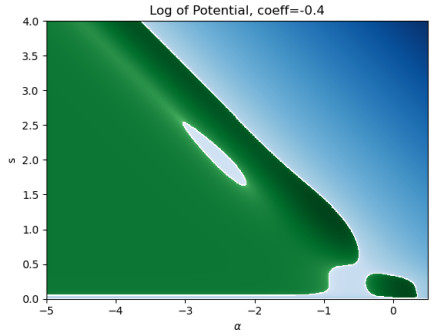

Figure 2.3: Log of Potential, c = -0.4

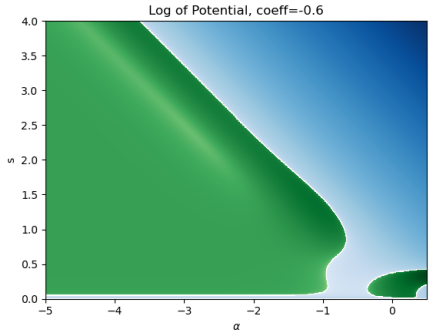

Figure 2.4: Log of Potential, c = -0.6

term ($c = 0$). We can define left and right barriers approximated in Ref [1] to characterize trajectories of particles in this potential. These barriers are shown in Figure 2.5 and are approximated by

$$\alpha_{\text{max}}(s) \approx \ln(2k/5) - s \tag{2.33}$$

$$\alpha_{left}(s) \approx \ln(k/2) - s \tag{2.34}$$

and

$$\alpha_{\text{right}}(s) \approx \ln(2\gamma^2) - s. \tag{2.35}$$

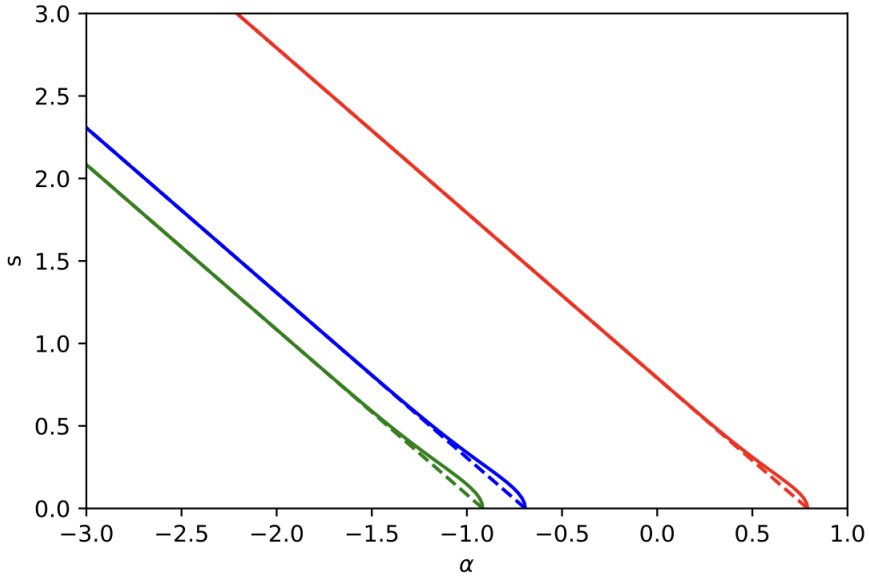

Figure 2.5: Approximate left and right barries of potential. Adapted from Bojowald and Peterson [1].

Trajectories that cross the green line, α_{left} are considered tunneling trajectories. Additionally, we call the region between α_{left} and the red line, α_{right}, a channel. Trajectories that start in the channel will often remain there as s increases and α decreases.

Figures 2.3-2.4 show the logarithm of the potential with varying negative coefficients. The green regions represent negative potential and the blue regions represent positive potential. Trajectories start with zero total energy so they are restricted to the green region. As the coefficient becomes more negative, the channel becomes less pronounced. It is not clear to define a region that trajectories can tunnel out of. Therefore, the focus of analysis is on positive values for the coefficient.

Figures 2.6-2.9 show the logarithmic of the potential with varying positive coefficients. As the coefficient increases, it is no longer possible for trajectories that start near the origin to travel up the channel. Trajectories that grow to large s within the channel are not well approximated by our constraint. Therefore, we restrict the tunnelling analysis to the region around the origin where $-1.5 < \alpha < 0$ and $0 < s < 1.5$.

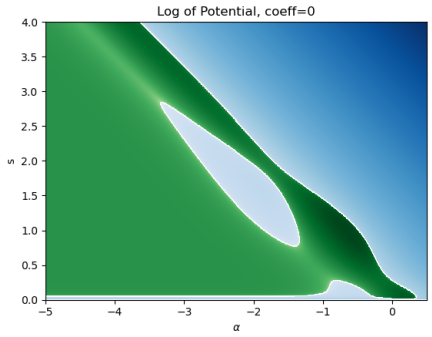

Figure 2.6: Log of Potential, c = 0.0

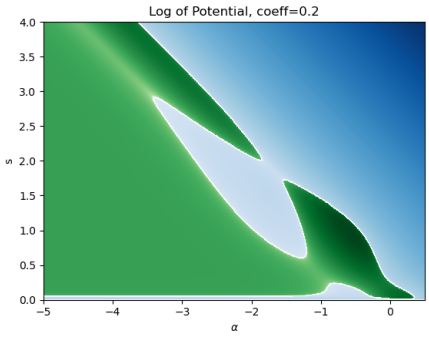

Figure 2.7: Log of Potential, c = 0.2

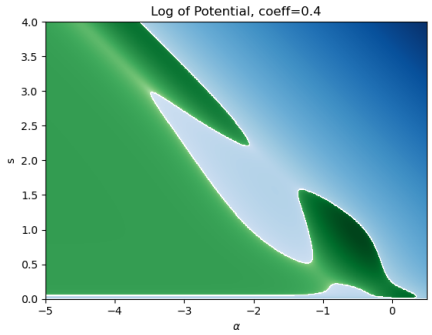

Figure 2.8: Log of Potential, c = 0.4

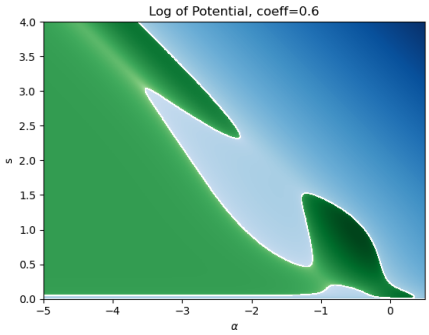

Figure 2.9: Log of Potential, c = 0.6

Trajectories

3.1 Tunneling Trajectories

We analyze the trajectories resulting from the potential with different coefficients for the cubic term. This is done with python by numerically solving the Hamiltonian. The particles have zero energy so the initial constraint is that $H = 0$. The Hamiltonian is given by

$$H = p_\alpha^2 + p_s^2 + \frac{U}{s^2} + \frac{1}{2}\left(U_p(\alpha + s) + U_p(\alpha - s)\right) + \frac{1}{12}U^{''''}s^4 + cU^{'''}s^3 \tag{3.36}$$

and we numerically solve for the trajectories using Hamilton's equations,

$$\dot{\alpha} = \frac{\partial H(\alpha, p_\alpha, s, p_s)}{p_\alpha} \tag{3.37}$$

$$\dot{p_\alpha} = \frac{\partial H(\alpha, p_\alpha, s, p_s)}{\alpha} \tag{3.38}$$

$$\dot{s} = \frac{\partial H(\alpha, p_\alpha, s, p_s)}{p_s} \tag{3.39}$$

and

$$\dot{p_s} = \frac{\partial H(\alpha, p_\alpha, s, p_s)}{s}. \tag{3.40}$$

Figures 3.10-3.15 show the resulting trajectories with initial conditions $\alpha = -0.1$, $p_\alpha = 0$, and $s = 0.1$, and varying coefficients c. We obtain the initial value of p_s by solving $H = 0$ with the other three initial variables. The python code used to produce these trajectories is supplied

in appendix A. For the trajectories in Figures 3.10-3.16, p_s varies slightly but is approximately 2.5. The orange line shows the trajectory in positive time and the blue line shows the trajectory in negative time. When $c = 0.1$, the trajectory escapes in negative time and when $c = 0.3$, the trajectory escapes in both positive and negative time.

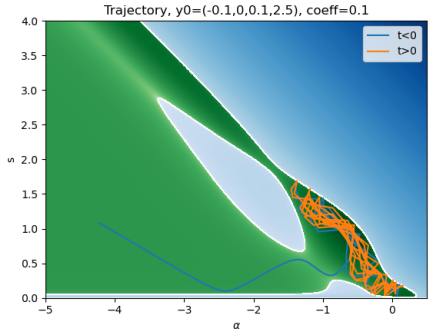

Figure 3.10: Trajectory with $y_0 = (-0.1, 0, 0.1, 2.5)$, $c = 0.1$

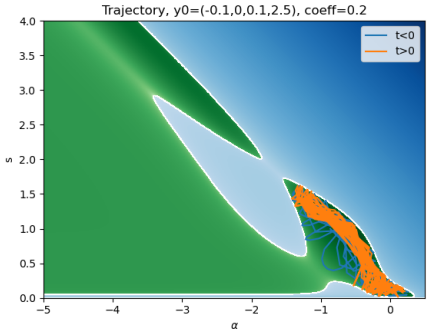

Figure 3.11: Trajectory with $y_0 = (-0.1, 0, 0.1, 2.5)$, $c = 0.2$

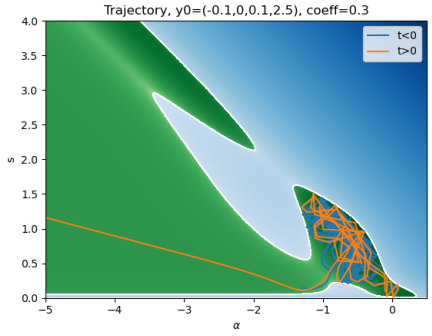

Figure 3.12: Trajectory with $y_0 = (-0.1, 0, 0.1, 2.5)$, $c = 0.3$

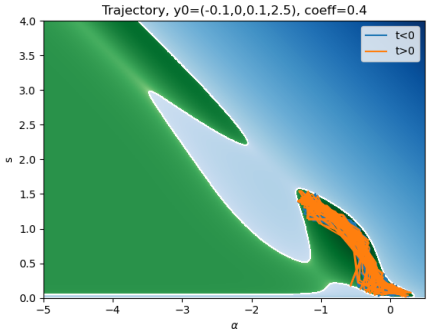

Figure 3.13: Trajectory with $y_0 = (-0.1, 0, 0.1, 2.5)$, $c = 0.4$

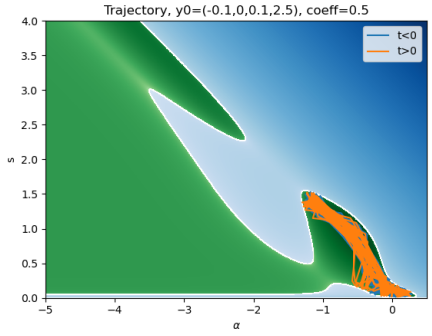

Figure 3.14: Trajectory with $y_0 = (-0.1, 0, 0.1, 2.5)$, $c = 0.5$

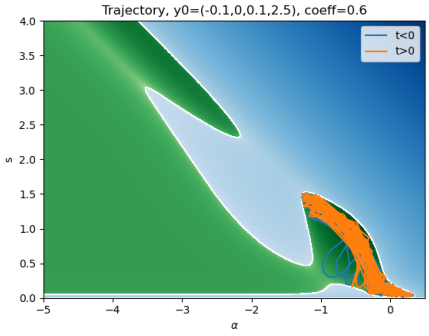

Figure 3.15: Trajectory with $y_0 = (-0.1, 0, 0.1, 2.5)$, $c = 0.6$

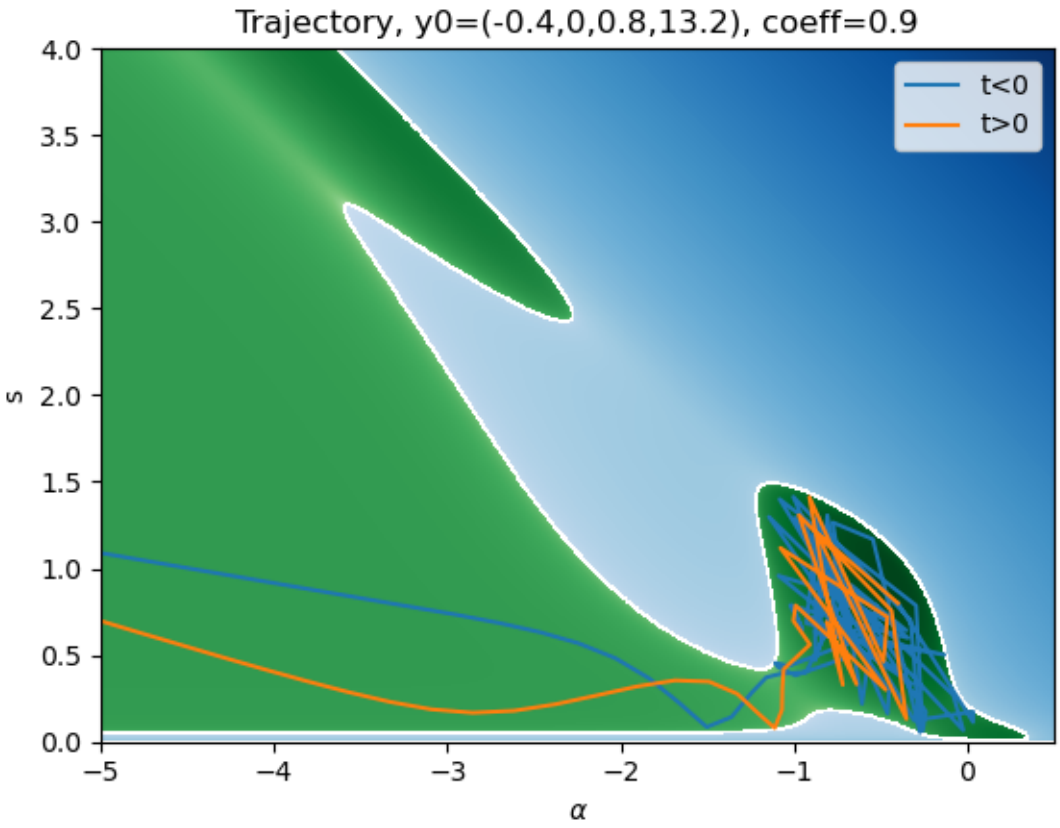

Figure 3.16: Trajectory with
$y_0 = (-0.4, 0, 0.8, 13.2)$, $c = 0.9$

The trajectory shown in Figure 3.16 is an example of one that escapes in both negative and positive time. As the particle exits the channel bound by α_{left}, its trajectory is largely unchanged and it acts like a free particle.

There is no clear way to predict if the trajectory will escape or not. The qualitative outcome of the trajectory is extremely sensitive to the initial conditions and this chaotic behavior is the focus of discussion in the next chapter. We discuss how the probability of tunneling is affected by the coefficient of the cubic term.

Chaotic Behavior

4.1 Lattice Points

To study the chaotic behavior of the trajectories and their sensitivity to initial conditions, we take a lattice of initial condition points and characterize the trajectories qualitatively. A trajectory escapes if it crosses to the left of the green line approximation, α_{left}. The trajectories that don't escape are red, those that escape in either positive or negative time are blue, and those that escape in both positive and negative time are green. The code for this algorithm is provided in appendix B.

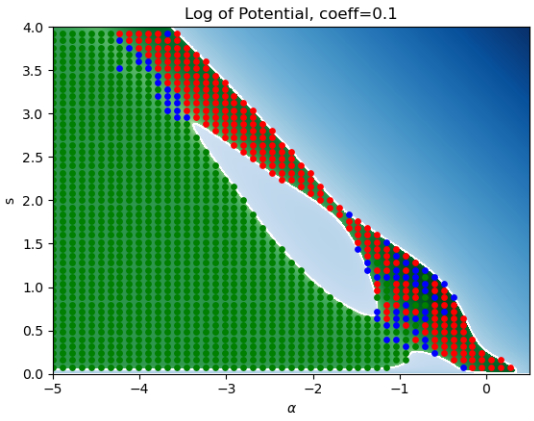

Figure 4.17: Lattice Points, $c = 0.1$

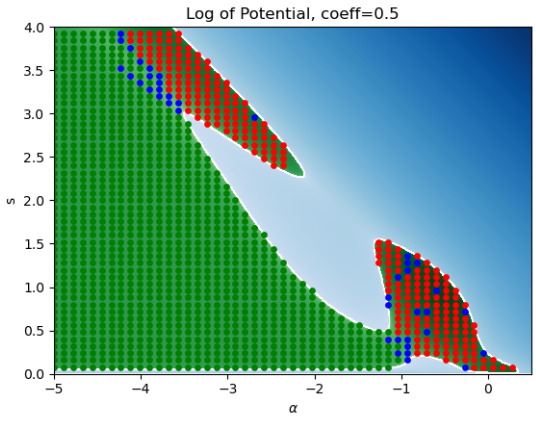

Figure 4.18: Lattice Points, $c = 0.5$

Figures 4.17 and 4.18 show the characterization of lattice points for coefficients of 0.1 and 0.5 respectively. In the area around $-2 < \alpha < 0$ and $0 < s < 2$, the outcome of the trajectories is sensitive to initial conditions and trajectories will randomly tunnel out.

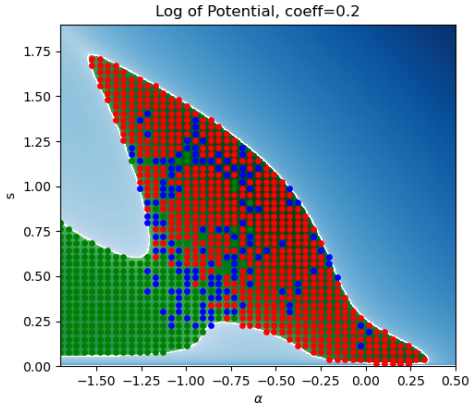

Figure 4.19: Lattice Points, $c = 0.2$

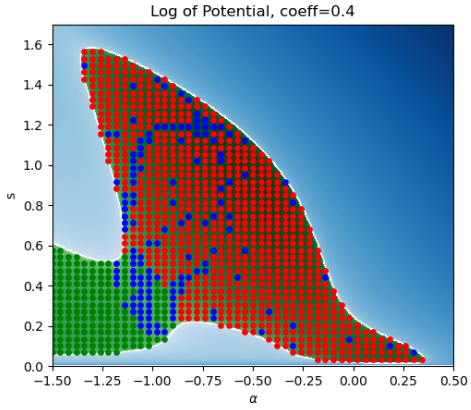

Figure 4.20: Lattice Points, $c = 0.4$

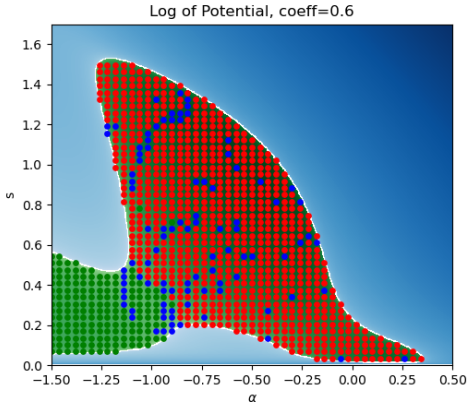

Figure 4.21: Lattice Points, $c = 0.6$

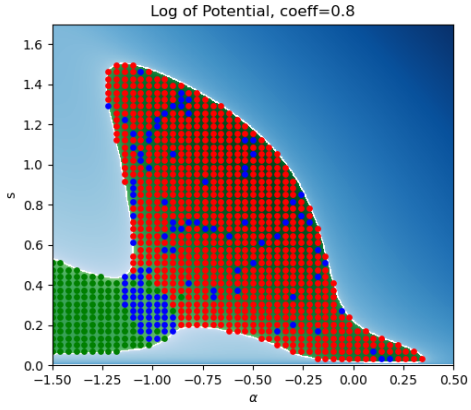

Figure 4.22: Lattice Points, $c = 0.8$

Figures 4.19-4.22 show the characterization of points in the region of interest for different coefficients. We can compare the fraction of points within the channel that tunnel for different coefficients, given in table 4.1.

Coefficient	Red	Blue	Green
0.1	0.6413	0.2448	0.1139
0.2	0.7600	0.1724	0.0676
0.4	0.8565	0.1297	0.0138
0.6	0.8877	0.0959	0.0164
0.8	0.8964	0.0965	0.0071
1.0	0.9133	0.0843	0.0024

Table 4.1: Fraction of Trajectories that Tunnel

We observe that a smaller coefficient for the potential term corresponds to a higher fraction, or

probability, of tunneling outside of the region. This is consistent with a widened trapped region and smaller gap to escape for the larger coefficient shown in the potential in Figure 4.19 compared to the smaller coefficient shown in Figure 4.22. Note that the absolute value of the probability of escaping is likely dependent on time steps and end time used in the numerical integration of the Hamiltonian. For this particular analysis, we chose 100 time steps and ended at time $t = 50$. A longer end time would likely correlate to a higher fraction of escaping trajectories. However, the relative probability between two different coefficients is still a good indication of tunneling probabilities because the same value for the time steps and end time were used throughout the numerical computations.

Conclusion

The addition of a cubic term to our approximation of the semi-classical cosmological model exhibits chaotic behavior in the trajectories of zero-energy particles. As the coefficient of the cubic term increases, the negative region of the potential changes shape and the resulting trajectories change. Various trajectories were studied and their outcomes were qualitatively classified.

With approximately the same initial point, varying the coefficient of the cubic term changed the outcome in an unpredictable way. Because the system is so sensitive to initial conditions, changing the coefficient changed the initial condition very slightly to maintain the $H = 0$ constraint and therefore varied the trajectory drastically.

The most direct way to analyze the chaotic nature of the trajectories is through classifying the tunneling of a lattice of initial points. This was done by defining three types of trajectories: trajectories that escape to the left, trajectories that escape to the right, and trajectories that do not escape. There was large variability in the outcome of trajectories starting from certain areas of the potential and a finer lattice was probed in the area of interest. Within the finer lattice, varying outcomes were also exhibited, suggesting that the true pattern is fractal.

The main result is observing how the cubic term changes the potential, the resulting trajectories, and the chaotic nature of the system. A larger coefficient for the cubic term cooresponds to a lower probability of tunneling. Additionally, the chaotic nature of the system suggests that fractal behavior and a fractal dimension can be observed, as calculated for another potential by Cornish and Levin [5]. The analysis could be furthered by using a finer lattice and calculating this fractal dimension.

Bibliography

[1] Martin Bojowald and Pip Petersen. Tunneling dynamics of an oscillating universe model. *Journal of Cosmology and Astroparticle Physics*, 2022(05):007, may 2022.

[2] Martin Bojowald. Canonical description of quantum dynamics. *Journal of Physics A: Mathematical and Theoretical*, 55(50):504006, dec 2022.

[3] James B. Hartle. Gravity: An introduction to einstein's general relativity. *American Journal of Physics*, Oct 2003.

[4] Bekir Bayta, Martin Bojowald, and Sean Crowe. Canonical tunneling time in ionization experiments. *Physical Review A*, 98(6), dec 2018.

[5] Neil J. Cornish and Janna J. Levin. Mixmaster universe: A chaotic farey tale. *Physical Review D*, 55(12):7489–7510, jun 1997.

Appendix A

Python Code for Trajectories

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import math
from sympy import *
from matplotlib import cm
import argparse
import pickle
import time
start_time = time.time()

'''
Initial values and coefficent are parsed.
Goes for negative and positive times

Creates three figures:
1. trajs_... Subplots of a, pa, s, ps vs t
2. traj_... plots trajectory, s vs a
3. potent_... log of potential '''

parser = argparse.ArgumentParser()
```

```
parser.add_argument('a0', type=str)
parser.add_argument('pa0', type=str)
parser.add_argument('s0', type=str)
parser.add_argument('coeff', type=str)
args = parser.parse_args()

timesteps = 200
end_time = 50
pixels = 500

t = np.linspace(0,end_time,timesteps)
t_plot = np.linspace(-end_time,end_time,2*timesteps)
# note that graph is slightly off: the value at t=0 occurs twice

ln = False # Create ln plot of potential?
pickles = False # Pickle dump ln plot?
traj = True # Create plot of traj? (s vs alpha)
trajs = False # Create plot of trajs? (a, pa, s, ps vs t)
overlay = True # Should traj plot be overlayed with potential?
########################################

x = Symbol('x')
a = Symbol('a')
pa = Symbol('pa')
s = Symbol('s')
ps = Symbol('ps')

var = [x, a, pa, s, ps]

def Up(x):
    b= 0.1
    k=1.0
    g= 1.05
```

```
    p=1.0
    var = [x]
    Up = exp(4*x)/b**2*(k-2*exp(x)+exp(2*x)/g**2)-p**2
    return Up

def H(a,pa,s,ps,coeff):
    var = [a,pa,s,ps]
    U = 10**(-2)/4
    return pa**2+ps**2+U/(s**2)+(1/2)*(Up(a+s)+Up(a-s))+1/12*diff(diff(diff(
   diff(Up(a),a),a),a),a)*s**4+coeff*diff(diff(diff(Up(a),a),a),a)*s**3

def ps0_solve(a0,pa0,s0,coeff):
    var = [x]
    U = 10**(-2)/4
    Up_plus = Up(x).subs(x,a0+s0)
    Up_minus = Up(x).subs(x,a0-s0)
    U_ppp = diff(diff(diff(Up(x),x),x),x).subs(x,a0)
    U_pppp = diff(diff(diff(diff(Up(x),x),x),x),x).subs(x,a0)
    return (-(pa0**2+U/s0**2+(1/2)*(Up_plus+Up_minus)+1/12*U_pppp*s0**4+coeff*
   U_ppp*s0**3))**(1/2)

def PotPlot(anum,snum,coeff):
    var = [x]
    U = 10**(-2)/4
    Up_plus = Up(x).subs(x,anum+snum)
    Up_minus = Up(x).subs(x,anum-snum)
    U_ppp = diff(diff(diff(Up(x),x),x),x).subs(x,anum)
    U_pppp = diff(diff(diff(diff(Up(x),x),x),x),x).subs(x,anum)
    H = U/snum**2+(1/2)*(Up_plus+Up_minus)+1/12*U_pppp*snum**4+coeff*U_ppp*
   snum**3
    return H

def potent(y,t):
```

```
    var = [a,pa,s,ps]
#Determines the canonical equations
    adot = expand(diff(H(a,pa,s,ps,coeff),pa))
    padot = expand(-diff(H(a,pa,s,ps,coeff),a))
    sdot = expand(diff(H(a,pa,s,ps,coeff),ps))
    psdot = expand(-diff(H(a,pa,s,ps,coeff),s))
#Creates an array for each of the solved canonical variable outputs
    for i in range(4):
        adot = adot.subs(var[i],y[i])
        padot = padot.subs(var[i],y[i])
        sdot = sdot.subs(var[i],y[i])
        psdot = psdot.subs(var[i],y[i])
    return[adot,padot, sdot, psdot]

coeff_str = args.coeff
coeff = float(args.coeff)
a0 = float(args.a0)
pa0 = float(args.pa0)
s0 = float(args.s0)
ps0 = float(ps0_solve(a0,pa0,s0,coeff)) #solve for ps0 by setting H=0
y0str = np.array([args.a0,args.pa0,args.s0,str(round(ps0,1))])
y0=  np.array([a0,pa0,s0,ps0])  #initial conditions (a,pa,s,ps)
y0_neg = np.array([a0,-pa0,s0,-ps0])

print('begin cubic.py')

savedata = '('+y0str[0]+','+y0str[1]+','+y0str[2]+','+y0str[3]+')_coeff=' +
   coeff_str
print('savedata: '+savedata)


if traj or trajs:
    print('begin odeint for y0')
```

```
    y = odeint(potent,y0,t)
    print('begin odeint for y0_neg')
    y_neg = odeint(potent,y0_neg,t)
    print('end odeint')

    a= np.flip(y_neg[:,0],0)
    pa= np.flip(y_neg[:,1],0)
    s= np.flip(y_neg[:,2],0)
    ps= np.flip(y_neg[:,3],0)

    a=np.append(a,y[:,0])
    pa=np.append(pa,y[:,1])
    s=np.append(s,y[:,2])
    ps=np.append(ps,y[:,3])

if trajs:
    fig,axs=plt.subplots(6)
    fig.suptitle('Trajectories, y0=('+y0str[0]+','+y0str[1]+','+y0str[2]+','+
   y0str[3]+'), coeff=' +str(coeff))
    fig.set_figheight(20)
    fig.set_figwidth(10)
    axs[0].set_title("a", fontsize=20)
    axs[1].set_title("pa", fontsize=20)
    axs[2].set_title("s", fontsize=20)
    axs[3].set_title("ps", fontsize=20)
    axs[4].set_title("Potential", fontsize=20)
    axs[5].set_title("H", fontsize=20)
    plt.xlabel("time", fontsize=10)
    plt.ylabel("   ", fontsize=10,verticalalignment='center',y=2.5)

    W_plot = []
    H_plot = []
    for i in range(len(a)):
```

```
149         W_plot.append(PotPlot(a[i],s[i],coeff))
150         H_plot.append(PotPlot(a[i],s[i],coeff)+pa[i]**2+ps[i]**2)
151 
152 
153     axs[0].plot(t_plot,a)
154     axs[1].plot(t_plot,pa)
155     axs[2].plot(t_plot,s)
156     axs[3].plot(t_plot,ps)
157     axs[4].plot(t_plot,W_plot)
158     axs[5].plot(t_plot,H_plot)
159 
160     fig.savefig('trajs_'+savedata+ '.png')
161     print('saved fig: '+ 'trajs_'+savedata+ '.png')
162     plt.clf()
163 
164 if traj:
165     fig = plt.figure()
166     ax = plt.axes()
167     ax.plot(a[0:int(len(a)/2)], s[0:int(len(s)/2)], label='t<0')
168     ax.plot(a[int(len(a)/2):len(a)], s[int(len(s)/2):len(s)], label = 't>0')
169     ax.legend()
170     plt.xlabel(r'$\alpha$')
171     plt.ylabel('s')
172     plt.title('Trajectory, y0=('+y0str[0]+','+y0str[1]+','+y0str[2]+','+y0str
    [3]+'), coeff=' +str(coeff))
173     if overlay:
174         bounds = [np.min(a),np.max(a),np.min(s),np.max(s)]
175         #anum = np.linspace(bounds[0]-0.1*abs(np.min(a)-np.max(a)),bounds
    [1]+0.1*abs(np.min(a)-np.max(a)),pixels)
176         #snum = np.linspace(bounds[2]-0.1*abs(np.min(s)-np.max(s)),bounds
    [3]+0.1*abs(np.min(s)-np.max(s)),pixels)
177         with open('vars/anum_'+str(coeff), 'rb') as f:
178             anum = pickle.load(f)
```

```
        with open('vars/snum_'+str(coeff), 'rb') as f:
            snum = pickle.load(f)
        with open('vars/p_'+str(coeff), 'rb') as f:
            pos = pickle.load(f)
        with open('vars/n_'+str(coeff), 'rb') as f:
            neg = pickle.load(f)
        ### If potential is not stored in file: ###
        '''anum = np.linspace(-5,0.5,pixels)
        snum = np.linspace(0,4,pixels)
        pos = np.empty((len(anum),len(snum)))
        neg = np.empty((len(anum),len(snum)))
        pos[:] = np.NaN
        neg[:] = np.NaN
        for p in range(len(anum)):
            for m in range(len(snum)):
                value = PotPlot(anum[p],snum[m],coeff)
                #print("value: ",value)
                #print(math.isnan(value))
                if not math.isnan(value):
                    if value > 0:
                        pos[-m-1,p]= math.log(value)
                    if value<0:
                        neg[-m-1,p]= math.log(-value)'''
        snum, anum = np.meshgrid(snum, anum)

        plt.imshow(pos, extent =[anum.min(), anum.max(), snum.min(), snum.max
    ()], cmap='Blues')
        plt.imshow(neg, extent =[anum.min(), anum.max(), snum.min(), snum.max
    ()], cmap='Greens')
        fig.savefig('traj_y0='+savedata+ '.png')
    else:
        fig.savefig('traj_y0='+savedata+ '.png')
    print('saved fig: '+'traj_y0='+savedata+ '.png')
```

```
    plt.clf()


#ln of potential and plot
############################################################
if ln:
    print('begin ln')
    if coeff == 0.1:
        amin = -1.8
        smax = 1.9
    elif coeff == 0.2:
        amin = -1.7
        smax = 1.9
    elif coeff == 0.3:
        amin = -1.6
        smax = 1.8
    elif coeff >= 0.4:
        amin = -1.5
        smax = 1.7
    else:
        print('coeff error: ', coeff)
    anum = np.linspace(amin,0.5,pixels)
    snum = np.linspace(0,smax,pixels)

    pos = np.empty((len(anum),len(snum)))
    neg = np.empty((len(anum),len(snum)))
    pos[:] = np.NaN
    neg[:] = np.NaN
    for i, p in enumerate(range(len(anum))):
        print(str(i)+' out of '+str(pixels))
        for m in range(len(snum)):
            value = PotPlot(anum[p],snum[m],coeff)
            if value > 0:
```

```
243                 pos[-m-1,p]= math.log(value)
244             if value<0:
245                 neg[-m-1,p]= math.log(-value)
246     if pickles:
247         print('begin pickle dumping')
248         with open('vars/p_'+str(coeff), 'wb') as f:
249             pickle.dump(pos,f)
250         with open('vars/n_'+str(coeff),'wb') as f:
251             pickle.dump(neg,f)
252         with open('vars/anum_'+str(coeff), 'wb') as f:
253             pickle.dump(anum,f)
254         with open('vars/snum_'+str(coeff), 'wb') as f:
255             pickle.dump(snum,f)
256     snum, anum = np.meshgrid(snum, anum)
257 
258     fig = plt.figure()
259     ax=plt.axes()
260     ax.set_title('Log of Potential, coeff='+coeff_str)
261     ax.imshow(pos, extent =[anum.min(), anum.max(), snum.min(), snum.max()],
    cmap='Blues')
262     ax.imshow(neg, extent =[anum.min(), anum.max(), snum.min(), snum.max()],
    cmap='Greens')
263     ax.set_xlabel(r'$\alpha$')
264     ax.set_ylabel('s')
265     fig.savefig('potent_'+str(coeff)+'.png')
266     print('saved fig: '+ 'potent_'+str(coeff)+'.png')
267 
268 print("complete in "+str(round((time.time() - start_time)/60,2))+" min")
269 
270 ############################################################
```

Appendix B

Python Code for Lattice Plot

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import math
from sympy import *
from sympy import Symbol
from sympy import pi
from sympy import exp
import time
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('points', type=int)
parser.add_argument('tsteps', type=int)
parser.add_argument('tend', type=int)
args = parser.parse_args()

start_time = time.time()
tsteps = args.tsteps
points = args.points
tend = args.tend
```

```
22
23 coeff = 0.1
24 t = np.linspace(0,tend,tsteps)
25 outfilename = 'lr_points'+str(points)+'_tsteps'+str(tsteps)+'_tend'+ str(tend)
       +'_coeff'+str(coeff)+'.png'
26 print(outfilename)
27 ### Characterizes initial conditions for a lattice of points: no escape,
       escape to the left, escape to the right ###
28 ########################################
29
30 x = Symbol('x')
31 a = Symbol('a')
32 pa = Symbol('pa')
33 s = Symbol('s')
34 ps = Symbol('ps')
35
36 var = [x, a, pa, s, ps]
37 k = 1.0
38 g = 1.05
39
40 def Up(x):
41     b= 0.1
42     k=1.0
43     g= 1.05
44     p=1.0
45     var = [x]
46     Up = exp(4*x)/b**2*(k-2*exp(x)+exp(2*x)/g**2)-p**2
47     return Up
48
49 def H(a,pa,s,ps,coeff):
50     var = [a,pa,s,ps]
51     U = 10**(-2)/4
52     return pa**2+ps**2+U/s**2+(1/2)*(Up(a+s)+Up(a-s))+1/12*diff(diff(diff(diff
```

```
    (Up(a),a),a),a),a)*s**4+coeff*diff(diff(diff(Up(a),a),a),a)*s**3

def ps0_solve(a0,pa0,s0,coeff):
    var = [x]
    U = 10**(-2)/4
    Up_plus = Up(x).subs(x,a0+s0)
    Up_minus = Up(x).subs(x,a0-s0)
    U_ppp = diff(diff(diff(Up(x),x),x),x).subs(x,a0)
    U_pppp = diff(diff(diff(diff(Up(x),x),x),x),x).subs(x,a0)
    return (-(pa0**2+U/s0**2+(1/2)*(Up_plus+Up_minus)+1/12*U_pppp*s0**4+coeff*
   U_ppp*s0**3))**(1/2)

def PotPlot(anum,snum,coeff):
    var = [x]
    U = 10**(-2)/4
    Up_plus = Up(x).subs(x,anum+snum)
    Up_minus = Up(x).subs(x,anum-snum)
    U_ppp = diff(diff(diff(Up(x),x),x),x).subs(x,anum)
    U_pppp = diff(diff(diff(diff(Up(x),x),x),x),x).subs(x,anum)
    W = U/snum**2+(1/2)*(Up_plus+Up_minus)+1/12*U_pppp*snum**4+coeff*U_ppp*
   snum**3
    return W

def potent(y,t):
    var = [a,pa,s,ps]
#Determines the canonical equations
    adot = expand(diff(H(a,pa,s,ps,coeff),pa))
    padot = expand(-diff(H(a,pa,s,ps,coeff),a))
    sdot = expand(diff(H(a,pa,s,ps,coeff),ps))
    psdot = expand(-diff(H(a,pa,s,ps,coeff),s))
#Creates an array for each of the solved canonical variable outputs
    for i in range(4):
        adot = adot.subs(var[i],y[i])
```

```
        padot = padot.subs(var[i],y[i])
        sdot = sdot.subs(var[i],y[i])
        psdot = psdot.subs(var[i],y[i])
    return[adot,padot, sdot, psdot]

#solve for ps0 by setting H=0
a_range = np.linspace(-1.5,0,points)
s_range = np.linspace(0,1.5,points)
ps_vals = []
panum = 0

print('begin fractal_lr.py')

a_list = []
pa_list = []
s_list = []
ps_list = []
aneg_list = []
paneg_list = []
sneg_list = []
psneg_list = []
grid = np.zeros((len(s_range),len(a_range)))
grid[:] = np.NaN
color_list = ['Red', 'Green', 'Blue', 'Purple']
count = 0
fig = plt.figure()

for a_index in range(len(a_range)):
    for s_index in range(len(s_range)):
        anum = a_range[a_index]
        snum = s_range[s_index]
        ps0 = ps0_solve(anum,panum,snum,coeff)
        if type(ps0) != Mul:
```

```
116             psnum = float(ps0)
117             ps_vals.append(psnum)
118             y0=  np.array([anum,panum,snum,psnum])  #initial conditions (a,pa,
    s,ps)
119             y0neg = np.array([anum, -panum, snum, -psnum])
120             y = odeint(potent,y0,t)
121             yneg = odeint(potent,y0neg,t)
122
123             a_list.append(y[:,0])
124             pa_list.append(y[:,1])
125             s_list.append(y[:,2])
126             ps_list.append(y[:,3])
127             aneg_list.append(yneg[:,0])
128             paneg_list.append(yneg[:,1])
129             sneg_list.append(yneg[:,2])
130             psneg_list.append(yneg[:,3])
131
132             VALUE = 0 #value of 0 (Red) is no escape; 1 (green) is left, 2 (
    blue) is right
133             if y[-1,0] < math.log(k/2)-y[-1,2] or yneg[-1,0] < math.log(k/2)-
    yneg[-1,2]:
134                 VALUE = 1
135             elif y[-1,0] > math.log(2*g**2)-y[-1,2] or yneg[-1,0] > math.log
    (2*g**2)-yneg[-1,2]:
136                 VALUE = 2
137             grid[a_index,s_index] = VALUE
138             plt.scatter(anum,snum, color=color_list[VALUE])
139             count +=1
140             print(str(count) + '/'+str(points**2)+ ' completed in ' + str(
    round((time.time() - start_time)/60,2)) +' min: ('+str(anum)+','+str(panum
    )+','+str(snum)+ ','+str(round(ps0,1))+'), value='+str(VALUE))
141         else:
142             count+=1
```

```
143             print(str(count) + '/'+str(points**2)+ ' is imaginary: ('+str(anum
    )+','+str(panum)+','+str(snum)+str(ps0)+')')
144
145 bounds = [np.min(a_range),np.max(a_range),np.min(s_range),np.max(s_range)]
146 anum = np.linspace(bounds[0]-0.1*abs(np.min(a_range)-np.max(a_range)),bounds
    [1]+0.1*abs(np.min(a_range)-np.max(a_range)),100)
147 snum = np.linspace(bounds[2]-0.1*abs(np.min(s_range)-np.max(s_range)),bounds
    [3]+0.1*abs(np.min(s_range)-np.max(s_range)),100)
148
149 pos = np.empty((len(anum),len(snum)))
150 neg = np.empty((len(anum),len(snum)))
151 pos[:] = np.NaN
152 neg[:] = np.NaN
153 for p in range(len(anum)):
154     for m in range(len(snum)):
155         value = PotPlot(anum[p],snum[m],coeff)
156         if value > 0:
157             pos[-m-1,p]= math.log(value)
158         if value<0:
159             neg[-m-1,p]= math.log(-value)
160 snum, anum = np.meshgrid(snum, anum)
161 plt.imshow(pos, extent =[anum.min(), anum.max(), snum.min(), snum.max()], cmap
    ='Blues')
162 plt.imshow(neg, extent =[anum.min(), anum.max(), snum.min(), snum.max()], cmap
    ='Greens')
163 plt.title('Characterization of Trajectories')
164 fig.savefig(outfilename+'.png')
```

Maria Lan Bressan

EDUCATION

The Pennsylvania State University, Schreyer Honors College **University Park, PA**
Eberly College of Science Expected Graduation: *May 2023*
B.S. in Physics, Minor in Mathematics

RESEARCH EXPERIENCE

Pennsylvania State University **University Park, PA**
Senior Thesis Research *January 2022 – Current*

- Investigating chaos in quantum cosmological models under Martin Bojowald from Penn State's Department of Physics.
- Numerically solving hamiltonians with Python and MATLAB to study dynamics.

U.S. State Department CERN Program **Meyrin, Switzerland**
Research Intern *September 2022 – December 2022*

- Analyzing QCD Instantons with the ATLAS collaboration under Tancredi Carli to either measure or set limits on its cross section.
- Studying the applications of neural networks and algorithms that identify b-hadrons at low momentum to the search.

Cornell University **Ithaca, NY**
Research Experience for Undergraduates *June 2022 – August 2022*

- Searching for long-lived particles at the Large Hadron Collider with Ritchie Patterson's group.
- Understanding efficiencies of new vertex reconstruction algorithms by utilizing CERN's ROOT framework with Monte Carlo simulated CMS data.

Pennsylvania State University **University Park, PA**
Research Intern *January 2021 – May 2022*

- Investigated a method to predict the color of a chlorophyll-like structure on an exoplanet under Suvrath Mahadevan and Joe Ninan from the Department of Astronomy & Astrophysics.
- Synthesized knowledge in Astrophysics and Biology to simulate the absorbance of chlorophyll given a spectral irradiance input using Python.

LEADERSHIP

Pennsylvania State University **University Park, PA**
Learning Assistant *August 2020 - December 2021*

- Learning Assistant for Introductory Mechanics, Electricity & Magnetism, Fluids and Thermal Physics, and Wave Motion and Quantum Physics.
- Facilitated learning in a classroom of approximately 50 students by guiding them through in-class learning activities and holding office hours.

WORK EXPERIENCE

Cozy Thai Bistro **State College, PA**
Shift Manager *November 2018 - July 2021*

- Supervised a staff of 10 workers including scheduling and training. Maintained high standards of customer service and assisted guests by addressing questions and complaints directly.

TECHNICAL SKILLS

MATLAB, Python, LaTeX, Mathematica, Excel, ROOT, C++

HONORS AND ORGANIZATIONS

John and Elizabeth Holmes Teas fellowship
Society of Physics Students
Caltech FUTURE Ignited 2022
Sigma Pi Sigma
Phi Beta Kappa