AMATH 271: Final Project

Omer Sipra, Kamalesh Reddy

December 9, 2020

Abstract

The following project falls in the mathematical domain of Calculus of Variation. It exhibits a computer program that is designed to provide a path of least "insert independent variable here" for any problem in Theoretical Mechanics. The computer program uses principles from Lagrangian Mechanics to achieve this. The problemâĂŹs lagrangian (specifically, it's arguments) is the input to the program and, using the Euler-Lagrange equation, the programs provide the user with a plot of the minimized quantity vs the independent variable that they defined.

Introduction

The problem that inspired this Project is one of the oldest problems in Calculus of Variations. The problem was a challenge by the mathematician Johann Bernoulli. He described the problem as: âĂIJGiven two points A and B in a vertical plane, what is the curve traced out by a particle under the influence of gravity (constant), which starts at A and reaches B in the shortest time.âĂİ

In other words: 'A' is any arbitrary point (x_0, y_0) in 2 dimensional space and B is defined as (x_1, y_1) where $y_1 \leq y_0$ and $x_0 \neq x_1$. This results in B being a translation of A which is under (but not directly under A). The system includes a particle which is supposed to âĂIJfallâĂİ from A to B because of a constant gravitational force. The objective of the problem is to find the path that would require the least time for the particle to get to B.

The problem was attempted and successfully solved by several mathematicians. Isaac Newton famously solved the problem overnight. The solution of this problem was named the âĂIJBrachistochroneâĂİ. Johann BernoulliâĂŹs solution being the first, identified this brachistochrone to be the common cycloid.

A cycloid is the path traced out by a point on the circumference of a circle as it rolls on a horizontal surface without slipping.

Different solutions to this problem were developed until as early as the 2010's. One of these solutions is what inspired this project. It was developed by Mark Levi [2] in 2014. His solution shows this symmetry in the path of least time called the âĂIJtautochroneâĂİ property of a brachistochrone, which means that the path a particle would take to fall from any point on the cycloid to B would all be the same! This made us realize that for any mechanical problem there exists 1 solution that describes the path of a least quantity which can be scaled to any magnitude of the problemâĂŹs dimensions.

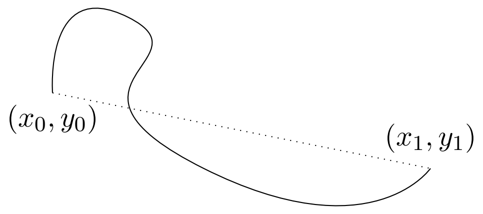
Another solution to the brachistochrone problem is provided in the textbook [1]. It involves finding the Lagrangian of the problem and using the Euler-Lagrange equation to find a DE that can be plotted. The resulting plot, as expected, reveals a brachistochrone.

If this method works for the brachistochrone problem, then it should work for any arbitrary problem in mechanics where a path can be traced. We hence decided to create a program to find the path of least time/distance of any problem that involves a Lagrangian. To achieve this we used the solutions mentioned above to design a program written in Python that can take a Lagrangian input (which corresponds to a mechanics problem of course) and then said code will show you the path of least distance/time. To demonstrate this project we tested the program for a couple of problems including problems from Mini-Project 2, AMATH 271 and problems from chapter 6 of Taylor.

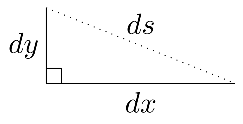
1

Taylor Example 6.1: Shortest Path Between 2 Points (warm-up)

1.1 Equations of problems



Now, consider a small section of the curve, ds (an arbitrarily small ds is a line):



We can hence write ds in terms of small steps in the x and y direction using the Pythagorean theorem:

$$ds^2 = dx^2 + dy^2 \implies ds = \sqrt{dx^2 + dy^2} \implies ds = \sqrt{1 + \left(\frac{dy}{dx}\right)^2}\, dx \implies ds = \sqrt{1 + y'(x)^2}\, dx$$

Integrating on both sides from (x_0, y_0) to (x_1, y_1):
$\displaystyle\int_{(x_0,y_0)}^{(x_1,y_1)} ds = \int_{x_0}^{x_1} \sqrt{1 + \dot{y}^2}\, dx.$

1.2 Analysis

Our goal is to minimize the integral, $\displaystyle\int_{(x_0,y_0)}^{(x_1,y_1)} ds$. To minimize this, observe that, equivalently, we can take the integrand, $\sqrt{1+\dot{y}^2}$, to be our Lagrangian (the proof of which is omitted)[1]. This is the essence of Lagrangian mechanics, it says nature is an optimization problem and depicts “nature is lazy" in equations. We have,

$$\mathcal{L}(\dot{y}, y, x) = \sqrt{1 + \dot{y}^2}$$

[1]Look into "Calculus of variations", Euler-Lagrange equation", and "Principle of stationary action".

1.3 Solutions - *SPBTP.py*

Now, on to solving this problem! Now that we have the Lagrangian, we can feed it to our Python program while making necessary changes (passing parameters (here, constants), variables 'names', and the Lagrangian itself as parameters).
The mechanism of this program will be explained in detail for this warm-up problem:

1. We import the required libraries; in this program, we will only need `sympy` to output the derivative. However, we will not use it to solve the derivative, since most derivatives cannot be solved numerically, which is how sympy solves them. We will use a second program to solve the derivative numerically.
2. Declare the function, `givemeDE`. The function does not explicitly take the lagrangian as an argument (for reasons that I will state later); BUT, the lagrangian is defined locally in the very next step, this makes it easier for the user to define the lagrangian, which will be required either way.
3. This problem's lagrangian does not require us to define any parameter (clearly).
4. The user must define the lagrangian, `L`, manually.
   Note: Make sure the Lagrangian is a function of ***three*** *variables.*
5. We then assign each variable; the dependent variable's derivative, the dependent variable, and the independent variable; with strings. This displays the differential equation with appropriate strings in place of `dvarp`, `dvar`, and `ivar` (the general variables our program uses).
6. The next step is where Python solves for each term of the lagrangian and `prints` it to the console.
7. Then, it plugs the two terms into the Euler-Lagrange equation (defined implicitly) and isolates all terms.
8. Finally, it prints the `DE`. The final differential equation will be of the form $f(\dot{q}, q, t) = 0$.
9. Keep the format of the function `givemeDE` in mind.

Crux: the program take the lagrangian as "input" and gives us the DE: $f(\ddot{q}, \dot{q}, q, t) = 0$. This DE will now be solved, again, using Python. Running the code below (`derog.py`) as is, we get:

```
∂ℒ/∂y      = 0


∂ℒ/∂y'     = Derivative(y(x), x)/sqrt(Derivative(y(x), x)**2 + 1)


d/dx(∂ℒ/∂y') = Derivative(y(x), (x, 2))/sqrt(Derivative(y(x), x)**2 + 1) -
    Derivative(y(x), x)**2*Derivative(y(x), (x, 2))/(Derivative(y(x), x)**2 + 1)**(3/2)
Required DE: y''/(y'**2 + 1)**(3/2) = 0
```

`y''/(y'**2 + 1)**(3/2) = 0` $\implies$ $\ddot{y} = 0$: (This is, as expected, a straight line!)

$$y(x) = C_1 + C_2 x$$

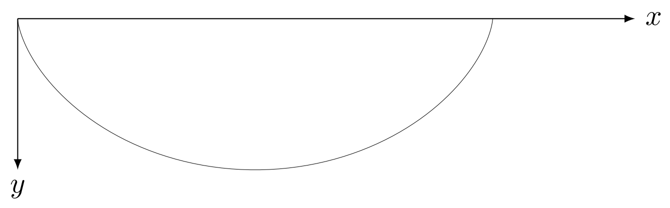
Python Code:

```
  # LIBRARIES
  import sympy as sp

  # FUNCTION
  def givemeDE(*args):

    #PARAMETERS

    #LAGRANGIAN
    def L(dvar, dvarp, ivar):
        return sp.sqrt(1 + dvarp**2)

    if len(args) == 3:
        # AESTHETICS
        def convertEq(x):
            x = x.replace("Eq(", "")
            x = x.replace(", 0)", "")
            x = x + " = 0"
            return x
        # AESTHETICS
        def convertDE(de):
            de_str = str(de)
            de_str = de_str.replace("Derivative(" + args[-2] + "(" + args[-1] + "), (" +
                args[-1] + ", 2))", args[-3] + "'" + "'")
            de_str = de_str.replace("Derivative(" + args[-2] + "(" + args[-1] + "), " +
                args[-1] + ")", args[-3] + "'")
            de_str = de_str.replace(args[-2] + "(" + args[-1] + ")", args[-3])
            return de_str

        # DEFINES ASSIGNED SYMBOLS
        ivar = sp.Symbol(args[-1])
        dvar = sp.Function(args[-2])(ivar)
        dvarp = sp.Derivative(dvar, ivar)

        # PRINTS EACH TERM OF THE LAGRANGIAN
        print("∂ℒ/∂" + args[-3] + "    =", sp.diff(L(dvar, dvarp, ivar), dvar))
        print('\n')
        print("∂ℒ/∂" + args[-3] + "'" + "   =", sp.diff(L(dvar, dvarp, ivar), dvarp))
        print('\n')
        print("d/d" + args[-1] + "(∂ℒ/∂" + args[-3] + "')" + " =",
            sp.diff(sp.diff(L(dvar, dvarp, ivar), dvarp), ivar))

        # FINDS DE USING TERMS ABOVE
        de = sp.simplify(sp.Eq(sp.diff(L(dvar, dvarp, ivar), dvar) -
            sp.diff(sp.diff(L(dvar, dvarp, ivar), dvarp), ivar), 0))

        # PRINTS OUT DE
        print("Required DE:", convertDE(convertEq(str(de))))

# EXECUTE PROGRAM!
givemeDE("y", "y", "x")
```

2

Taylor Example 6.2: Brachistochrone

2.1 Equations of problems



"Describe at least three equations for three different methods of solving this problem"

2.2 Analysis

"Analyse each equation and provide intuitive insight to explain why each method is significant."

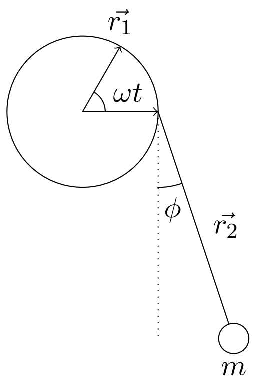
2.3 Solutions - *.py*

"Contrast between solutions if possible, which methos was the most interesting, why?"

3

Mini-Project 2: Question 1

3.1 Equations of problems



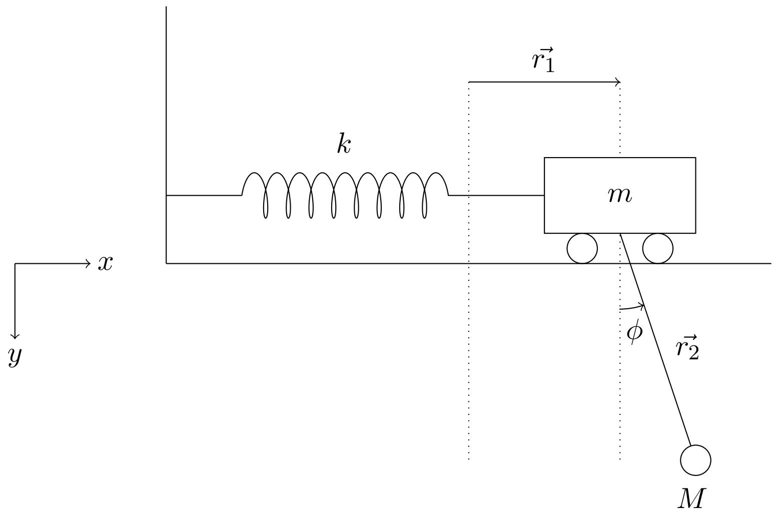
3.2 Analysis

3.3 Solutions - *.py*

4

Mini-Project 2: Question 1

4.1 Equations of problems



4.2 Analysis

4.3 Solutions - *.py*

1 Conclusions

References

[1] Taylor, J. R. *Classical Mechanics*. University Science Books, 2005.

[2] Mark Levi *Quick! Find a Solution to the Brachistochrone Problem*. Mathematical Curiosities, 2005.

[3] Rodrigo Matos Carnier *Energy-efficient optimal control of robotic gait by indirect methods*. Research Gate , 2017