Contents

§1 Lagrangian Mechanics

Let us start from defining the **action**:

$$S\left[\boldsymbol{q}\left(t\right)\right]=\int_0^T L\left(\boldsymbol{q},\dot{\boldsymbol{q}},t\right)\,dt,$$

the **Eular-Lagrange equation** is derived from $\delta S=0$, with an additional restriction $\delta\boldsymbol{q}\left(0\right)=\delta\boldsymbol{q}\left(T\right)=0$:

$$\begin{aligned}\delta\int_0^T L\,dt&=\int_0^T\left(\nabla_{\boldsymbol{q}}L\cdot\delta\boldsymbol{q}+\nabla_{\dot{\boldsymbol{q}}}L\cdot\delta\dot{\boldsymbol{q}}\right)\,dt\\&=\int_0^T\left(\nabla_{\boldsymbol{q}}L-\frac{d}{dt}\nabla_{\dot{\boldsymbol{q}}}L\right)\cdot\delta\boldsymbol{q}\,dt=0.\end{aligned}$$

Since the choice of $\delta\boldsymbol{q}$ is arbitrary, we obtain the Eular-Lagrange equation (we place the highest order derivative at the beginning):

$$\frac{d}{dt}\boldsymbol{\nabla}_{\dot{\boldsymbol{q}}}L-\boldsymbol{\nabla}_{\boldsymbol{q}}L=\boldsymbol{0}. \tag{1}$$

But what is the **Lagrangian** function L?

†a Free Particle

First, there are some symmetry about the Lagrangian:

1) Space translation: $L\left(\boldsymbol{q},\dot{\boldsymbol{q}},t\right)=L\left(\dot{\boldsymbol{q}}\right)$.
2) Rotation: $L\left(\dot{\boldsymbol{q}}\right)=L\left(\left|\dot{\boldsymbol{q}}\right|^2\right)$.

In this sense:

$$2\frac{d}{dt}L'\left(\left|\dot{\boldsymbol{q}}\right|^2\right)\dot{\boldsymbol{q}}.$$

Compare to the Newtonian Mechanics:

$$m\ddot{\boldsymbol{q}}=0,$$

we make $L'\left(\left|\dot{\boldsymbol{q}}\right|^2\right)$ as a constant $m/2$:

$$L_{\text{free}}=T=\frac{1}{2}m\left|\dot{\boldsymbol{q}}\right|^2. \tag{2}$$

†b Conservative Force

In the case of conservative force:

$$m\ddot{\boldsymbol{r}}=\boldsymbol{F}=-\boldsymbol{\nabla}V\left(\boldsymbol{q}\right).$$

Then we have

$$\frac{d}{dt}\boldsymbol{\nabla}_{\dot{\boldsymbol{q}}}T-\boldsymbol{\nabla}_{\boldsymbol{q}}T+\boldsymbol{\nabla}V\left(\boldsymbol{q}\right)=\frac{d}{dt}\boldsymbol{\nabla}_{\dot{\boldsymbol{q}}}\left(T-V\right)-\boldsymbol{\nabla}_{\boldsymbol{q}}\left(T-V\right)=0.$$

So we define

$$L_{\text{conserve}}=T-V=\frac{1}{2}m\left|\dot{\boldsymbol{q}}\right|^2-V. \tag{3}$$

†c Constraint Force

Under the following conditions, we can use the **generalized coordinate**:

1) There is some holonomic constraint $f\left(\boldsymbol{q},t\right)$.
2) The constraint force satisfies $\boldsymbol{F}\cdot\delta\boldsymbol{q}=0$.

In Newtonian Mechanics:

$$m\ddot{\boldsymbol{q}}=\boldsymbol{F}_{\text{conserve}}+\boldsymbol{F}_{\text{constraint}}.$$

But in the variation of the action:

$$\begin{aligned}\delta S&=\int_0^T\left(m\ddot{\boldsymbol{q}}-\boldsymbol{F}_{\text{conserve}}-\boldsymbol{F}_{\text{constraint}}\right)\cdot\delta\boldsymbol{q}\,dt\\&=\int_0^T\left(m\ddot{\boldsymbol{q}}-\boldsymbol{F}_{\text{conserve}}\right)\cdot\delta\boldsymbol{q}\,dt=0\end{aligned}$$

Chapter 1

Introduction to Statistics

§1 Data Type

†a Object

Python is an **object-oriented** programing language. Everything is an **object** in Python:

$$\text{object} = \begin{cases} \text{identity}, \\ \text{type / class}, \\ \text{value / state}, \\ \text{methods / behaviors / operations}. \end{cases}$$

```
# print the identity, type, and the value for 4
print(id(4),type(4),4)
# type of any type is a type, the type itself is a type
print(type(type(4)))
print(type(type(type(4))))
```

```
140711773227544 <class 'int'> 4
<class 'type'>
<class 'type'>
```

- **Identity**: it guarantees that different objects have distinct identities at any given time.
- **Type**: objects of the same type support the same operations, and share the same properties.

†b Binding and Input

In Python, the **assignment** of $a = b$ is like making the name a pointing to the object b.

```
# an example for binding
a,b=4,print
print(type(a),a,type(b),b,
     id(a),id(4),id(b),id(print))
b(a+5,"hello")
```

```
<class 'int'> 4 <class 'builtin_function_or_method'>
    <built-in function print> 140723891816984
    140723891816984 2069908885472 2069908885472
9 hello
```

The basic input in Python is through the function input( ). The input takes ONE string as prompt, and it reads input as a string.

```
# an example for input function
n=input(f"{a} and hello\n")
print(type(n),n)
```

```
4 and hello
5
<class 'str'> 5
```

†c Numeric

The following are numeric types:

$$\text{bool} \subset \text{int} \subset_? \text{float} \subset_? \text{complex}$$

```
# an example for the above data types
print(type(True),True,type(1),1,
     type(1.0),1.0,type(1+0j),1+0j)
```

```
<class 'bool'> True <class 'int'> 1 <class 'float'> 1.0
    <class 'complex'> (1+0j)
```

```
# subset example
if True==1==1.0==1+0j:
   print("Yes")
else:
   print("No")
```

```
Yes
```

We can use bool( ), int( ), float( ), and complex( ) to convert a string to the corresponding data type from input( );

```
# input string to number
n=input("type in an integer\n")
print(type(n),n,type(int(n)),int(n))
```

```
type in an integer
17
<class 'str'> 17 <class 'int'> 17
```

identically map from a subset to a larger set, or canonically map from the supset to the restricted set:

```
# identical map and canonical map
print(int(False),float(5),int(3.7))
```

```
0 5.0 3
```

†d More on Bool

```
# logic and bool
print(type(1==0))
print(type(""),bool(""))
if not "":
    print("statement or bool value defined can be used in
        logic")
```

```
<class 'bool'>
<class 'str'> False
statement or bool value defined can be used in logic
```

For statements and numbers, there is a **special method** bool:

```
# special method __bool__()
print(type((5==3).__bool__()),(5==3).__bool__(),
     id((5==3).__bool__()),id(False))
```

```
<class 'bool'> False 140723890821168 140723890821168
```

```
# special method __bool__() for numbers
print(type((0+3.5j).__bool__()),(0+3.5j).__bool__(),
     id((0+3.5j).__bool__()),id(True))
```

```
<class 'bool'> True 140723890821136 140723890821136
```

For some data like string, list etc., the special method len is defined.

```
# sepcial method __len__() for strings
print(type("abcd".__len__()),"abcd".__len__(),
     id("abcd".__len__()),id(4))
```

```
<class 'int'> 4 140723891816984 140723891816984
```

The bool( ) has a **protocol**:

1) If special method bool is defined, then return.
2) Else if the special method is defined, then return True if len is not 0 and vise versa.
3) Else return True.

In general, bool( ) is used for logical determination.

†e More on Float

For float, there are some useful **regular methods**:

```
# regular method is_integer() for floats
print(type((1.3).is_integer()),(1.3).is_integer())
```

```
<class 'bool'> False
```

```
# regular method as_integer_ratio() for floats
print(type((0.5).as_integer_ratio()),
     (0.5).as_integer_ratio())
```

```
<class 'tuple'> (1, 2)
```

In fact, for binary, 0.1 has infinitly many number after the decimal point:

```
# decimal in binary
x=0.1
print(x,f"{x:.17f}")
```

```
0.1 0.10000000000000001
```

For actual finite decimal, it is better to use a **standard library** decimal, and a function in the libary, Decimal:

```
# standard library decimal
import decimal
print(type(decimal.Decimal("0.1")),"\n",
     f"{decimal.Decimal("0.1"):.50f}\n",
     f"{decimal.Decimal(0.1):.50f}")
```

```
<class 'decimal.Decimal'>
 0.10000000000000000000000000000000000000000000000000
 0.10000000000000000555111512312578270211815834045410
```

A better way is to use a standard library fractions, and a fucntion in the libary, Fraction; and we can check if 0.1 is indeed the 0.1:

```python
# standard library fractions
import fractions
print(type(fractions.Fraction(1,10)),"\n",
     fractions.Fraction(1,10),fractions.Fraction("0.1"),
     fractions.Fraction(1,10)==decimal.Decimal("0.1"))
```

```
<class 'fractions.Fraction'>
 1/10 1/10 True
```

Bibliography