

Contents

§1. Lagrangian Mechanics		
†a	Free Particle	1
†b	Conservative Force	1
†c	Constraint Force	1

I INTRODUCTION TO STATISTICS

§1. Data Type		
†a	Object	3
†b	Binding and Input	3
†c	Numeric	3
†d	More on Bool	4
†e	More on Float	4

§1 Lagrangian Mechanics

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

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

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

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

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

$$\frac{d}{dt} \nabla_{\dot{\mathbf{q}}} L - \nabla_{\mathbf{q}} L = \mathbf{0}. \quad (1)$$

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

†a Free Particle

First, there are some symmetry about the Lagrangian:

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

In this sense:

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

Compare to the Newtonian Mechanics:

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

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

$$L_{\text{free}} = T = \frac{1}{2} m |\dot{\mathbf{q}}|^2. \quad (2)$$

†b Conservative Force

In the case of conservative force:

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

Then we have

$$\frac{d}{dt} \nabla_{\dot{\mathbf{q}}} T - \nabla_{\mathbf{q}} T + \nabla V(\mathbf{q}) = \frac{d}{dt} \nabla_{\dot{\mathbf{q}}} (T - V) - \nabla_{\mathbf{q}} (T - V) = 0.$$

So we define

$$L_{\text{conserve}} = T - V = \frac{1}{2} m |\dot{\mathbf{q}}|^2 - V. \quad (3)$$

†c Constraint Force

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

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

In Newtonian Mechanics:

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

But in the variation of the action:

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

Chapter 1

Introduction to Statistics

§1 Data Type

†a Object

Python is an **object-oriented** programming 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

```
# 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
```

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

0 5.0 3

```
# 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
```

```
# 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
```

```
# special method __len__() for strings
print(type("abcd").__len__(), "abcd".__len__(),
```

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

- 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.

```
# 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)
```

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

0.1 0.100000000000000000000001

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

```
<class 'decimal.Decimal'>  
0.10000000000000000000000000000000000000000000000000000000000000  
0.100000000000000000555111512312578270211815834045410
```

Bibliography