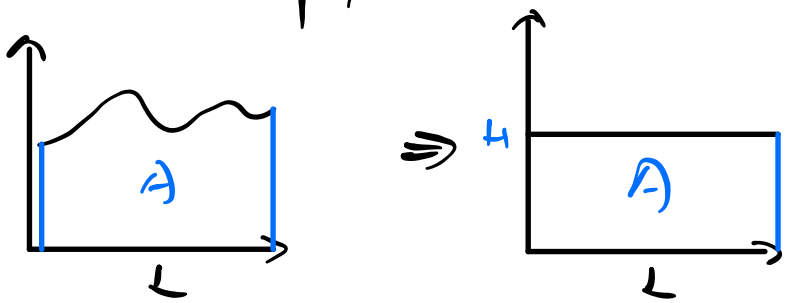
1. Monte Carlo.

随机抽样 / 统计模拟.



H = Average [Sum(曲线的y值)]

2. Newton-Cotes.

scipy.integrate.newton_cotes(*rn, equal=0*) [source]

Return weights and error coefficient for Newton-Cotes integration.

Suppose we have (N+1) samples of f at the positions x_0, x_1, ..., x_N. Then an N-point Newton-Cotes formula for the integral between x_0 and x_N is:

$$\int_{x_0}^{x_N} f(x)dx = \Delta x \sum_{i=0}^{N} a_i f(x_i) + B_N (\Delta x)^{N+2} f^{N+1}(\xi)$$

where $\xi \in [x_0, x_N]$ and $\Delta x = \frac{x_N - x_0}{N}$ is the average samples spacing.

If the samples are equally-spaced and N is even, then the error term is $B_N (\Delta x)^{N+3} f^{N+2}(\xi)$.

Parameters: **rn** : *int*

The integer order for equally-spaced data or the relative positions of the samples with the first sample at 0 and the last at N, where N+1 is the length of *rn*. N is the order of the Newton-Cotes integration.

equal : *int, optional*

Set to 1 to enforce equally spaced data.

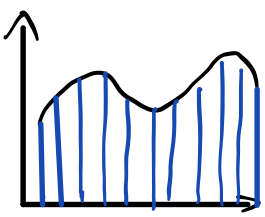
Returns: **an** : *ndarray*

1-D array of weights to apply to the function at the provided sample positions.

B : *float*

Error coefficient.

返回值注意一下.



Spacing : N

$\int f(x)\,dx =$

$dx \sum w_i \cdot f(x_i) + \text{Error}$

Spacing.

每一部分的权重

3. Simpson / Gauss (Fixed)

scipy.integrate.simpson(*y, x=None, dx=1.0, axis=-1, even='avg'*) [source]

Integrate y(x) using samples along the given axis and the composite Simpson's rule. If x is None, spacing of dx is assumed.

If there are an even number of samples, N, then there are an odd number of intervals (N-1), but Simpson's rule requires an even number of intervals. The parameter 'even' controls how this is handled.

$$\int_{x_1}^{x_3} f(x)dx = h\left[\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{1}{3}f_3\right] + O(h^5 f^{(4)})$$

3 points – 2 intervals in the integration

y - 要集成的阵列.
x - 采样 y 的点
dx - 间距

even : *str {'avg', 'first', 'last'}, optional*

'avg' : *Average two results:1) use the first N-2 intervals with* a trapezoidal rule on the last interval and 2) use the last N-2 intervals with a trapezoidal rule on the first interval.

'first' : *Use Simpson's rule for the first N-2 intervals with* a trapezoidal rule on the last interval.

'last' : *Use Simpson's rule for the last N-2 intervals with a* trapezoidal rule on the first interval.

Google:
对于间隔相等的奇数个样本. 若 Order ≤ 3.
结果是精确的.
若不相等, 则 order ≤ 2 才有精确结果.

实现:
```
x = np.arange(0, 10)
y = np.power(x, 3)
integrate.simpson(y, x)
```
→ 1642.5

分为 n 段.
每段由 Simpson 推出一个近似值, 再加在一起.
Simpson = 样形法外推.
↑ Accuracy.

scipy.integrate.fixed_quad(*func, a, b, args=(), n=5*) [source]

Compute a definite integral using fixed-order Gaussian quadrature.

Integrate *func* from *a* to *b* using Gaussian quadrature of order *n*.

Parameters:

func : *callable*
A Python function or method to integrate (must accept vector inputs). If integrating a vector-valued function, the returned array must have shape `(..., len(x))`.

a : *float*
Lower limit of integration.

b : *float*
Upper limit of integration.

args : *tuple, optional*
Extra arguments to pass to function, if any.

n : *int, optional*
Order of quadrature integration. Default is 5.

Returns:

val : *float*
Gaussian quadrature approximation to the integral

none : *None*
Statically returned value of None

⇒ 注意返回值. fixed_quad(y,x)[0].

4. BVP ODEs – Shooting Method.

两点边值

这里和 IVP 一起是比较方.

↓ 单点边值.

IVP:

↑ 看 IVP 会不会给 Method 解释.

`scipy.integrate.solve_ivp(fun, t_span, y0, method='RK45', t_eval=None, dense_output=False, events=None, vectorized=False, args=None, **options)` [source]

Solve an initial value problem for a system of ODEs.

This function numerically integrates a system of ordinary differential equations given an initial value:

```
dy / dt = f(t, y)
y(t0) = y0
```

Here t is a 1-D independent variable (time), y(t) is an N-D vector-valued function (state), and an N-D vector-valued function f(t, y) determines the differential equations. The goal is to find y(t) approximately satisfying the differential equations, given an initial value y(t0)=y0.

fun: right-hand side of the differential equation f(t,y).

↳ $\frac{dy}{dx}$

t_span:

Interval of integration (t0, tf). The solver starts with t=t0 and integrates until it reaches t=tf.

→ B.C [0, 20]

y(0) → y(20)

y0: an array specifying the initial state. → I.C [4]

y(0) = 4 y(20) = ?

Return:

t – 时间点 y – t处的解值.

---

shooting method.

考虑 BVP: $y'' = f(x, y, y')$ $a < x < b$

Given: $y(a) = \alpha$ $y(b) = \beta$ known

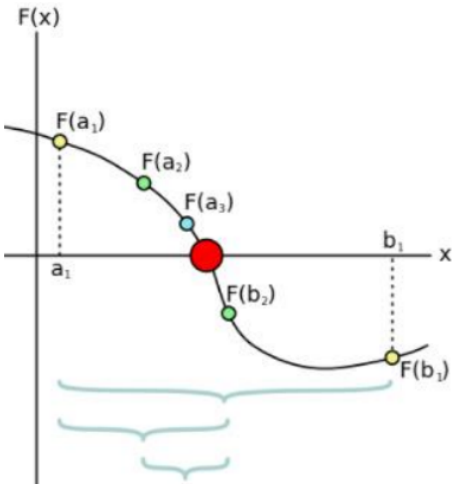
Assume: $y'(a) = t$ ← unknown

通过选取不同的 t 值，我们可以得到不同的 $y(b)$ 值.

当 $y(b) \to \beta$ 时，此时的 t 即为正确的初值.

J. Bisection / Newton-Raphson.

Bisection. 两点法.




```
#Bisection
def func(x):
    return x**2-1
print(op.bisect(func, 0, 2))    #0-2区间
```

Raphson

With a single starting point x_0

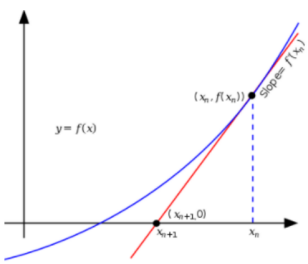
We know $f(x)$

So, we also know $f'(x_0)$

Find the intersection of the derivative line with the x axis

Use the intersection x coordinate as the starting point

Repeat the algorithm



$$f'(x_n) = \frac{f(x_n) - 0}{x_n - x_{n+1}} \Rightarrow x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Newton Raphson

Can be generalize to higher order derivatives – Householder's method

```
#Newton-Raphson
def f0(x):
    return x**3-1

def f1(x):                          #f0的导数
    return 3*x**2
print(op.newton(f0, 1.5,f1))        #估计0点在x=1.5处
```