

>

[base \(https://www.cnblogs.com/pkuoliver/archive/2010/10/06/1844725.html?utm_source=wechat_session&utm_medium=social&utm_oi=648897917608398848\)](https://www.cnblogs.com/pkuoliver/archive/2010/10/06/1844725.html?utm_source=wechat_session&utm_medium=social&utm_oi=648897917608398848)

[sotry-about-sqrt \(https://diducoder.com/sotry-about-sqrt.html\)](https://diducoder.com/sotry-about-sqrt.html)

[InvSqrt.pdf \(http://www.matrix67.com/data/InvSqrt.pdf\)](http://www.matrix67.com/data/InvSqrt.pdf)

二分法

```
double eps = 0.00000001;

float SqrtByBisection(float n) //用二分法
{
    if(n<0) //小于0的按照你需要的处理
        return n;
    float mid,last;
    float low,up;
    low=0;
    up=n;
    mid=(low+up)/2;
    do
    {
        if(mid*mid>n)
            up=mid;
        else
            low=mid;
        last=mid;
        mid=(up+low)/2;
    }while(abs(mid-last) > eps); //精度控制
    return mid;
}
```

```
In [29]: eps = 0.00000001
```

```
In [30]: def sqrt_by_bisection(n):
low, last = 0, 0
up = n
mid = (low + up)/2
while abs(mid - last) > eps:
    if mid * mid > n:
        up = mid
    else:
        low = mid
        last = mid
    mid = (up + low)/2
return mid
print(sqrt_by_bisection(2))

%timeit sqrt_by_bisection(2)
```

1.4142135605216026

6.26 μ s \pm 6.98 ns per loop (mean \pm std. dev. of 7 runs, 100000 loops each)

牛顿迭代法

```
double eps = 0.00000001;

float SqrtByNewton(float x)
{
    float val = x; //最终
    float last; //保存上一个计算的值
    do
    {
        last = val;
        val = (val + x/val) / 2;
    } while (abs(val - last) > eps);
    return val;
}
```

```
In [31]: def sqrt_by_newton(n):
        mid = n
        last = 0
        while abs(mid - last) > eps:
            last = mid
            mid = (mid + n/mid) / 2
        return mid
print(sqrt_by_newton(2))

%timeit sqrt_by_newton(2)

1.414213562373095
1.14 µs ± 0.357 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

InvSqrt

[Bit Twiddling Hacks \(http://graphics.stanford.edu/~seander/bithacks.html\)](http://graphics.stanford.edu/~seander/bithacks.html)

```
float InvSqrt(float x)
{
    float xhalf = 0.5f*x;
    int i = *(int*)&x; // get bits for floating VALUE
    i = 0x5f375a86 - (i >> 1); // gives initial guess y0
    x = *(float*)&i; // convert bits BACK to float
    x = x*(1.5f - xhalf*x*x); // Newton step, repeating increases accuracy
    x = x*(1.5f - xhalf*x*x); // Newton step, repeating increases accuracy

    return 1/x;
}
```

```
In [32]: # def inv_sqrt(n):
        #     xhalf = 0.5 * n
        #     i = n
        #     i = 0x5f375a86 - (i >> 1)
        #     x = i
        #     x = x * (1.5 - xhalf*x*x)
        #     x = x * (1.5 - xhalf*x*x)
        #     return 1/x
        # inv_sqrt(2)
```