>

> base (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)
>
> 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 μs ± 6.98 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

牛顿迭代法, 公式推导:

$$f(x) = x^2 - 2, 求f(x) = 0$$

$$x_1 = x_0 - \frac{f(x_0)}{f`(x_0)}$$

$$x_1 = x_0 - \frac{x_0^2 - 2}{2x_0} = (x_0 + 2/x_0)/2$$

```
# C
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)
```

SAS

```
%macro fcal(x=,t=,);
    data &t.;
    &t. = (&x.) ** 2 - 2;
    run;
%mend;

%macro fcal_x(b=,fl=0.000001);
    %let bf = 1;

    %do i = 0 %to 100;
        %fcal(x=&b., t=f);
        %fcal(x=&b.-&fl., t=f_0);
        %fcal(x=&b.+&fl., t=f_1);

        data f_all;
        merge f f_0 f_1;
        x0 = &b.;
        f_ = (f_1 - f_0)/(&fl.*2); # 导函数模拟
/*          f_ = 2 * x0; */
        x1 = x0 - f/f_;
        run;

        data _null_;
        set f_all;
        call symput("bb", x1);
        run;

        %fcal(x=&bb., t=f1);

        data _null_;
        set f1;
        f1_ = f1 < 0.01;
/*          f1_ = abs(&bb. - &b.) < 0.01; */
        call symput("brf", f1);
        call symput("bf", f1_);
        run;
        %put --------------------------- &bb. &brf. &bf.;

        %if &bf. = 1
            %then %let i = 100;
        %else %let b = &bb.;
    %end;
%mend;

%fcal_x(b=4);

# --------------------------- 2.2500000001 3.0625000005           0
# --------------------------- 1.5694444446 0.4631558647           0
# --------------------------- 1.4218903638 0.0217722067           0
# --------------------------- 1.4142342859 0.0000586154           1
```

InvSqrt

> Bit Twiddling Hacks (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)
#     return 1/x
# inv_sqrt(2)
```