快速平方根倒数算法 (Fast Inverse Square Root)

问题定义

如何快速求解 $\frac{1}{\sqrt{x}}$?

Background: IEEE 754

一个 32 bit 浮点数在 IEEE 754 标准中是由三个部分表征：1 bit 符号位，8 bit 指数位，23 bit 小数点位。

log2 Approximation

log2 实际上和 bit 表示方法有存在本质联系。给定一个浮点数 x，通过将这个数 bit 解释成为整数，再将这个整数做变换，可以得到 log2(x) 的近似值。

推导过程:

- 一个 32 bit 浮点数 x，8 bit 指数位为 E，23 bit 小数点位为 M，则一个正浮点数可以表示为

$x = (1 + \frac{M}{2^{23}}) \times 2^{E-127}$

对 x 取对数得到

$$\begin{aligned}\log_2 x &= \log\left(1 + \frac{M}{2^{23}}\right) + E - 127 \\ &\approx \frac{M}{2^{23}} + \mu + E - 127 \\ &\approx \frac{E \times 2^{23} + M}{2^{23}} - 127 + \mu\end{aligned}$$

其中，$E \times 2^{23} + M$ 正是一个浮点数解释成一个 32bit 整数后的值，如果记 $\hat{x} = E \times 2^{23} + M$，则

$\log_2 x \approx \frac{\hat{x}}{2^{23}} - 127 + \mu$

这里中间有一步利用了 $log(1+x) \approx x + \mu$，当，$log(1+x)$ 和 x 在 $x \in [0,1]$ 上的函数值比较接近，我们可以取合适的 μ 可以使整体的误差最小（这个值大概是 0.045 左右）。

Derivation

回到原问题：如何快速求解 $\frac{1}{\sqrt{x}}$？首先我们假定真实答案是 y，即有

$y = \frac{1}{\sqrt{x}}$

两边取对数

$\log_2 y = -\frac{1}{2}\log_2 x$

运用上面提到的 log2 近似，得

$\frac{\hat{y}}{2^{23}} - 127 + \mu = -\frac{1}{2} * (\frac{\hat{x}}{2^{23}} - 127 + \mu)$

一波化简后可以得到

$$\begin{aligned}\hat{y} &= \left(\frac{3}{2} \times 2^{23} \times (127 - \mu)\right) - \frac{1}{2}\hat{x} \\ &= K - \frac{1}{2}\hat{x}\end{aligned}$$

这里的 K 是一个常数，也正是源代码中的 magic number `0x5f3759df`。经过这一波操作，我们其实可以看到近似计算平方根倒数的方法很简单，对于输入 32bit 的浮点数 x：

1. 将 x 解释为一个 32 bit 的整数 $\hat{x}$
2. 对这个整数进行 scaling 和 shifting 操作，得到一个新整数 $\hat{y}$
3. 最后将其重新解释为浮点数 y，即为近似结果

Optimization

利用牛顿法迭代校准近似值，我们希望调整 y 的值使估计值和真实值的误差更小。即变量是 y，优化目标是误差值。

$y = \frac{1}{\sqrt{x}}$

Code

The original C code

```c
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                       // evil floating point bit level hacking
    i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//    y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed
    return y;
}
```

Python code below

```python
intpr2long = lambda n: struct.unpack(">l", struct.pack(">f", n))[0]
intpr2float = lambda n: struct.unpack(">f", struct.pack(">l", n))[0]

def fast_inv_sqrt(x):
    i = intpr2long(x)  # intepret as a long number
    i = 0x5f3759df - (i >> 1)  # scaling and shifting
    y = intpr2float(i)  # interpret as a float number
    y = y * (1.5 - (0.5 * x * y * y))  # 1 round of newton iteration
    return y
```