Министерство образования и науки Российской Федерации

**Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана»**

Кафедра «Прикладная математика»

Курсовая работа

по дисциплине «Математические пакеты»

Быстрое возведение в рациональную степень

Выполнил студент группы ФН12-51 *Чернов А.В.*

Научный руководитель к.ф.-м.н., доцент кафедры ФН-2 *Марчевский И.К.*

Москва — 2017

Оглавление

1. Стандартная реализация функции pow в языке golang

Рассмотрим реализацию функции pow в языке golang, опуская проверки на корректность входных данных.
Для начала заметим, что

$$x^y = \exp(y \log x) \tag{1}$$

К сожалению, погрешность вычисления функции $\exp x$ и $\log x$ в ЭВМ при больших x значительна, поэтому нельзя сразу применить эту формулу.
Перейдем к описанию алгоритма. Пусть x - это число, которое мы возводим в степень y. Если $y = |0.5|$, то возведение в степень выполняется через функцию sqrt, так как это значительно быстрее.

```
    case y == 0.5:
        return Sqrt(x)
    case y == -0.5:
        return 1 / Sqrt(x)
```

Далее берется модуль от числа y и выделятся целая и дробная часть числа $|y|$(функция Modf).

```
    absy := y
    flip := false
    if absy < 0 {
        absy = -absy
        flip = true
    }
    yi, yf := Modf(absy)
```

Формула (1) применяется для $yf \in (-0.5, 0.5]$

```
if yf != 0 {
    if yf > 0.5 {
        yf--
        yi++
    }
    a1 = Exp(yf * Log(x))
}
```

Далее используется алгоритм возведения в целую степень за время порядка $\log n$, где n — степень, в которую возводится число,с той лишь разницой, что в начале число x предстваляется, как: $x1 * 2^{xe}$(функция Frexp).

```
x1, xe := Frexp(x)
for i := int64(yi); i != 0; i >>= 1 {
    if i&1 == 1 {
        a1 *= x1
        ae += xe
    }
```

```
    x1 *= x1
    xe <<= 1
    if x1 < .5 {
        x1 += x1
        xe--
    }
}
```

Если $y < 0$, то полученный результат необходимо возвести в минус первую степень. Таким образом, мы получаем уже конечный результат:

```
if flip {
   a1 = 1 / a1
   ae = -ae
}
return Ldexp(a1, ae)
```

Где функция Ldexp обратная к Frexp.
Реализацию функции sqrt приводить не будем, но отметим что она работает значительно быстрее, поэтому если необходимо возвести в степень $n/2^m$ имеет смысл извлечь корень из числа m раз, а затем возвести сначала в степень n за время $\log n$. Данный алгоритм, представим уже на языке C++:

```
    template<typename T>
    static T quickPow2m(T x, int n, int m){
        T res=x;
        assert(m%2==0);
        while(m != 1){
            res=sqrt(res);
            m >>= 1;
       }
       return powNatural(res,n);
    }
```

2. Описание алгоритма быстрого возведения в степень

Вначале,напомним, что числа с плавающей точкой в ЭВМ хранится в следующем виде: $(-1)^S * M * B^E$,где S — знак, B — основание, E — порядок, а M — мантисса. Непосредственно в памяти ЭВМ хранится только знак порядок и мантисса. Например, в 32-битном числе знак занимает один бит, порядок 8 бит, мантисса 23 бита. Далее все выкладки приведены для 32-битного числа. Также отметим, что порядок в ЭВМ хранится со смещением, для 32-битного числа оно равно 127. Таким образом, целочисленная интерпретация числа имеет вид: $M + LE$, где $L = 2^{23}$. Пусть

$$m = \frac{M}{L}, \quad e = E - B, \tag{2}$$

где $B = 127$, тогда любое число можно представить в виде: $(1 + m)2^e$.
Далее рассмотрим уравнение вида:

$$y = x^p \tag{3}$$

и попытаемся его линеаризовать. Возьмем логарифм по основанию два от обеих частей уравнения:

$$\log_2 y = \frac{1}{p} \log_2 x$$

$$\log_2(1 + m_y) + e_y = \frac{1}{p}(\log_2(1 + m_x) + e_x)$$

Аппроксимируем $\log_2(1 + m)$ линейной функцией: $m + \sigma$, где $\sigma = 0.0450465$, так как $m \in [0, 1]$, то такая аппроксимация допустима, для наглядности изобразим график двух функций (рис. 1) и график модуля разности $|x + \sigma - \log_2(1 + m)|$ (рис. 2):

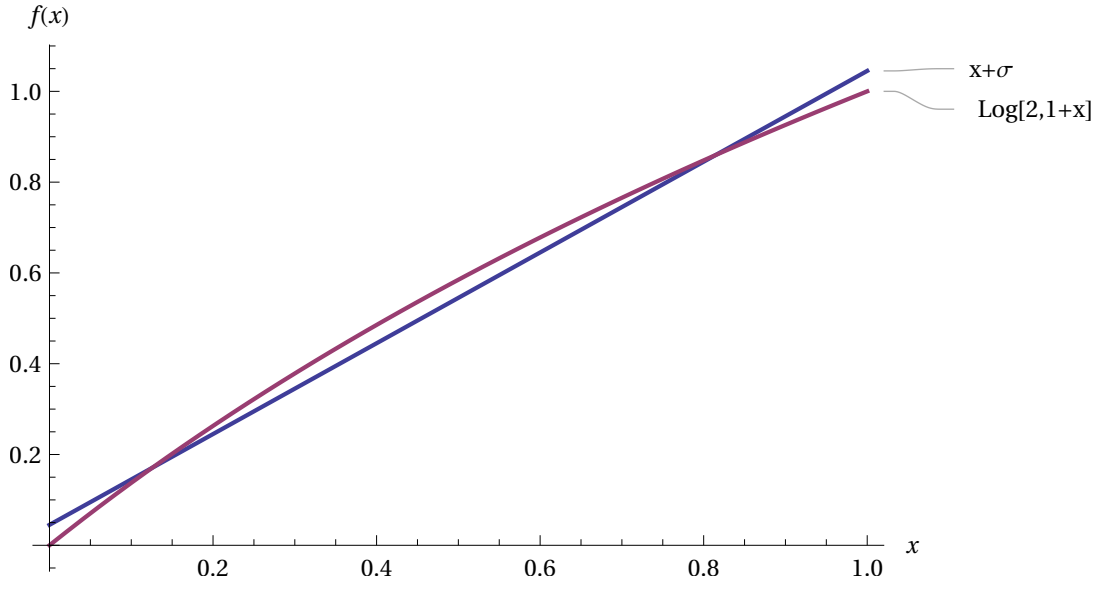

Рис. 1.

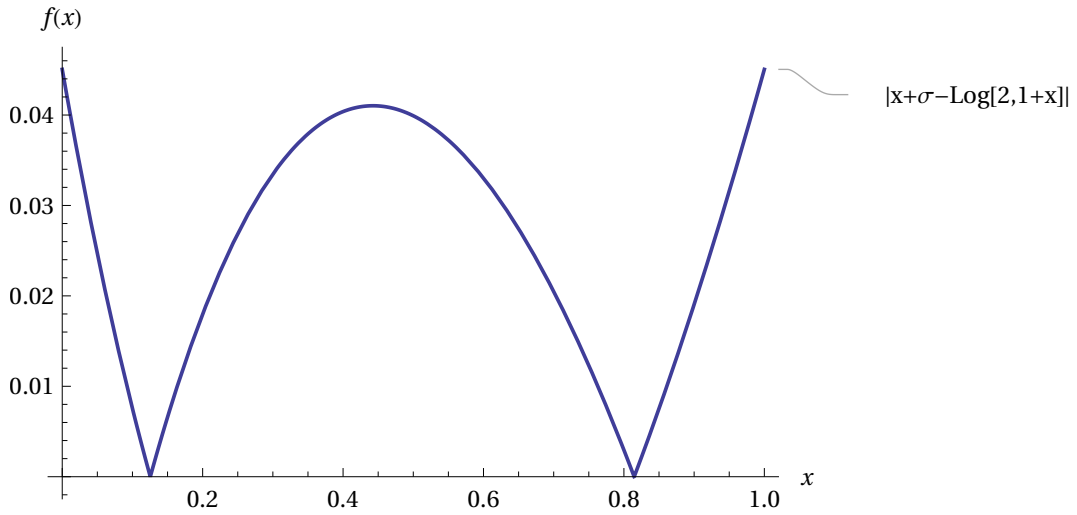

Рис. 2.

Воспользовавшись формулой (2), получим уравнение:

$$m_y + \sigma + e_y = p(m_x + \sigma + e_x)$$
$$M_y + LE_y = p(M_x + LE_x) + L(1-p)(E-\sigma)$$

$$M_y + LE_y = p(M_x + LE_x - L(E-\sigma)) + L(E-\sigma) \tag{4}$$

Заметим, что $M + LE$ является целочисленной интерпретацией числа, таким образом, мы получили формулу, которая позволяет возвести в любую степень действительную степень. Представим реализацию этого алгоритма на языке C++:

```
int i = *(int *) &x;
float p = 1.0f/q;
i = 1064975338 + p*(i - 1064975338);
float y = *(float *) &i;
```

К сожалению, погрешность данного алгоритма велика, поэтому результат вычисления, следует уточнить. В данной курсовой работе рассматривается метод Ньютона и его модификации.

3. Метод Ньютона

Рассмотрим нелинейное уравнение:

$$f(y) = y^{1/p} - x = 0 \tag{5}$$

И $k+1$ итерацию метода Ньютона:

$$y = y_k - \frac{f(y_k)}{f'(y_k)}.$$

где $f'(y) = \frac{1}{p}y^{1/p-1}$. После элементарных преобразований получаем уравнение:

$$y = y_k(1-p) - \frac{px}{y^{1/p-1}}$$

Таким образом, в общем случае, чтобы произвести итерацию Ньютона необходимо выполнить операцию возведения в дробную степень, поэтому имеет смысл посмотреть на задачу с другой стороны. Заметим, что аппроксимация решения методом Ньютона заметно упрощается, если $\frac{1}{p}$ целое число. Поэтому имеет смысл решать задачу возведения в рациональную степень, то есть $p = \frac{n}{m}$. Таким образом, сначала мы будем возводить число в целую степень n,это можно сделать за время $O(\log n)$, потом возводить в степень $\frac{1}{m}$. При таком подходе при аппроксимации решения методом Ньютона нам требуется всего-лишь возвести в целую степень. Для аппроксимации осуществим две итерации метода Ньютона:

```
float p1 = 1-p;
float px= p*x;
int q1 =q-1;
y = p1 * y + px/powNatural(y, q1);
return  p1* y + px/powNatural(y, q1);
```

4. Модификация метода Ньютона

Разложим по формуле Тейлора в о окрестности точки y^k с точностью до квадратичных слагаемых:

$$f(y^k) + f'(y^k)(y - y^k) + \frac{f''(y^k)}{2}(y - y^k)^2 \approx 0.$$

Решая квадратное уравнение относительно $(y - y^k)^2$, подставляя $f(y^k) = y^{1/p} - x; q = \frac{1}{p}$ и выбирая корень, который дает наилучшую аппроксимацию, получаем:

$$y = y^k + \frac{y(q - 2 + \sqrt{-1 + 2p + \frac{2(1-p)x}{y^q}}}{q-1}$$

На практике квадратный корень расскладывают по формуле Тейлора, но так как функция sqrt(x) хорошо оптимизирована это не дает существенного выигрыша в скорости, чтобы в этом убедиться мы реализуем оба алгоритма. С разложением корня:

```
float a = powNatural(y, q - 2);
float b = a * y;//powNatural(y,q-1)
float c = b * y;//powNatural(y,q)
float cx = c - x;
float d = ((q - 1) * a * cx) / (b * b * q);
return y - (cx) / (q * b) * (1 + 0.5f * d);
```

и без разложения:

```
return y *(q-2 + sqrtf(-1.0f+ 2.0f*( x*(1.0f-p)/powNatural(y,q) + p ))) / (q - 1);
```

5. Результаты

Будем возводить поэлементно массив в степень. Количество элементов в массиве равно 10^7. Тип данных float. Модификация метода Ньютона(a) — это метод, приведенный в предыдущей главе, без разложения корня по формуле Тейлора, а модификация метода Ньютона(b) с разложением.

Компилятор: gcc c++.

степень	метод	Отн. погр.	Время вып. отн powf
1/3	Метод Ньютона(1 итер.)	1.1e-03	0.40
	Метод Ньютона(2 итер.)	1.3e-06	0.55
	Модиф м. Ньютона(a)	6.96e-05	0.46
	Модиф. м. Ньютона(b)	6.15e-05	0.44
	Через sqrt	-	-
3/4	Метод Ньютона(1 итер.)	1.9e-03	0.47
	Метод Ньютона(2 итер.)	5.5e-06	0.61
	Модиф. м. Ньютона(a)	5.1e-05	0.56
	Модиф. м. Ньютона(b)	1.5e-04	0.58
	Через sqrt	2.6e-07	0.34
4/3	Метод Ньютона(1 итер.)	1.1e-03	0.44
	Метод Ньютона(2 итер.)	1.2e-06	0.60
	Модиф. м. Ньютона(a)	1.4e-05	0.52
	Модиф. м. Ньютона(b)	6.1e-05	0.49
	Через sqrt	-	-

Компилятор: MSVC.

степень	метод	Отн. погр.	Время вып. отн powf
1/3	Метод Ньютона(1 итер.)	9.0e-04	0.22
	Метод Ньютона(2 итер.)	1.2e-06	0.37
	Модиф м. Ньютона(a)	1.4e-05	0.49
	Модиф. м. Ньютона(b)	6.1e-05	0.26
	Через sqrt	-	-
3/4	Метод Ньютона(1 итер.)	1.9e-03	0.28
	Метод Ньютона(2 итер.)	5.4e-06	0.40
	Модиф. м. Ньютона(a)	5.13e-5	0.66
	Модиф. м. Ньютона(b)	1.5e-041	0.40
	Через sqrt	2.4e-07	0.38
4/3	Метод Ньютона(1 итер.)	9.0e-04	0.28
	Метод Ньютона(2 итер.)	7.6e-05	0.43
	Модиф. м. Ньютона(a)	1.01e-05	0.63
	Модиф. м. Ньютона(b)	4.5e-05	0.37
	Через sqrt	-	-

Комилятор Intel.

степень	метод	Отн. погр.	Время вып. отн powf
1/3	Метод Ньютона(1 итер.)	9.2e-04	0.42
	Метод Ньютона(2 итер.)	1.3e-06	0.73
	Модиф. м. Ньютона(a)	1.3e-05	0.73
	Модиф. м. Ньютона(b)	6.1e-05	1.45
	Через sqrt	-	-
3/4	Метод Ньютона(1 итер.)	5.7e-03	0.38
	Метод Ньютона(2 итер.)	5.4e-06	0.63
	Модиф. м. Ньютона(a)	5.1e-05	0.75
	Модиф. м. Ньютона(b)	1.5e-04	0.69
	Через sqrt	2.1e-07	0.42
4/3	Метод Ньютона(1 итер.)	9.0e-04	0.25
	Метод Ньютона(2 итер.)	6.7e-07	0.53
	Модиф. м. Ньютона(a)	1.02e-05	0.48
	Модиф. м. Ньютона(b)	4.5e-05	1.2
	Через sqrt	-	-

6. Заключение

В итоге, проанализировав все результаты, полученные в данной курсовой работе, можно сделать вывод, что возведение в рациональную степень можно выполнить за меньшее время, по сравнению со стандартной реализацией. Выбор метода зависит от задачи и от компилятора.

Список литературы

1. Галанин М.П., Савенков Е.Б. Методы численного анализа математических моделей. М.: МГТУ им. Н.Э. Баумана, 2010. 591 с.
2. Кормен Т.Х. Алгоритмы: Вводный курс. М.: Вильямс, 2015. 208 с.
3. Керниган Б.У., Донован А.А., Язык программирования Go. М.: Вильямс, 2016. 432 с.