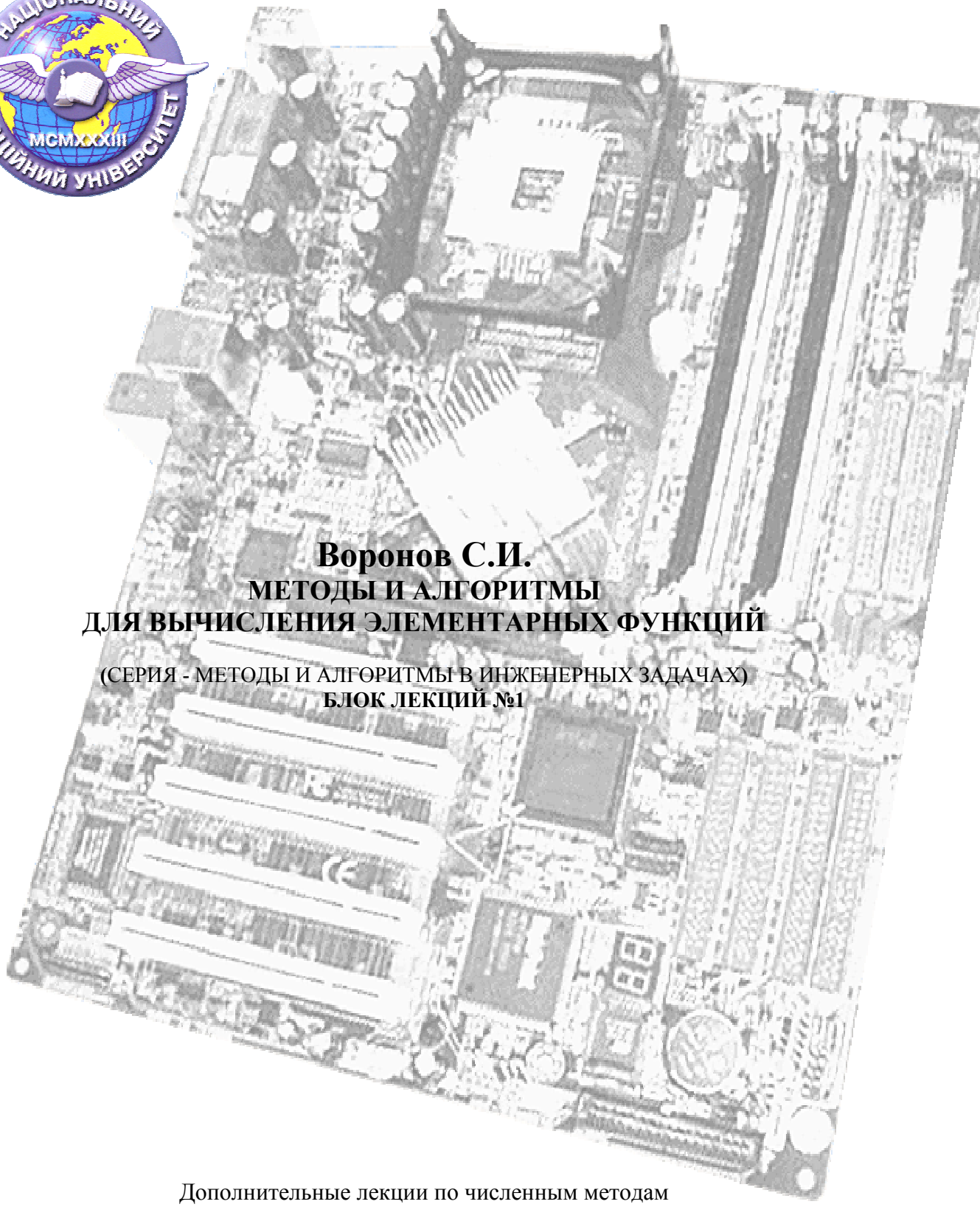
НАЦИОНАЛЬНЫЙ АВИАЦИОННЫЙ УНИВЕРСИТЕТ.



Воронов С.И.

МЕТОДЫ И АЛГОРИТМЫ ДЛЯ ВЫЧИСЛЕНИЯ ЭЛЕМЕНТАРНЫХ ФУНКЦИЙ

(СЕРИЯ - МЕТОДЫ И АЛГОРИТМЫ В ИНЖЕНЕРНЫХ ЗАДАЧАХ)

БЛОК ЛЕКЦИЙ №1

Дополнительные лекции по численным методам

(Редакция от 2015г.)

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Данное пособие предназначено для студентов системотехнических специальностей, как вспомогательный материал по дисциплинам «Вычислительная техника и алгоритмические языки» и «Программирование». Кроме того, пособие может использоваться в целом ряде других дисциплин, в которых рассматриваются вопросы нахождения аналитического вида характеристик разнообразных объектов (например, различных датчиков, передаточных функций и т.п.) или рассматриваются вопросы построения аналитического описания сигналов.

Поскольку дисциплины, которые используют методы и алгоритмы вычисления элементарных функций, преподаются в разные годы обучения студентов, то материал пособия излагается с опорой только на начальные сведения по высшей математике, которые включают основы дифференциального и интегрального исчисления в объеме первого года обучения.

Аппроксимация функций сходящимися степенными рядами.

1. Ряды Тейлора – Маклорена (определения)

Ряды Тейлора [1,2,3] являются степенными рядами, которые используются для аппроксимации различных функций, что в ряде случаев значительно упрощает их анализ и преобразование таких функций.

Традиционно ряд Тейлора определяют следующим образом:

$$f(x)=\sum_{k=0}^{\infty}\frac{f^{(k)}(a)}{k!}(x-a)^k$$

при этом предполагается, что функция $f(x)$ бесконечно дифферициируемая в окрстности точки a.

С математической точки зрения бесконечная длина такого ряда не является препятствием для его рассмотрения и дальнейших аналитических преобразований. С точки зрения реальных вычислений, приходится ограничиваться некоторой конечной длиной ряда. Ирыми словами, приходится ограничивать верхний индекс ряд Тейлора некоторым конечным значение N, что приводит ряд к следующему виду:

$$f(x)=\sum_{k=0}^{N}\frac{f^{(k)}(a)}{k!}(x-a)^k+R_N$$

где R_N называют остаточным членом, то есть, некоторым числом, которое отражает ошибку представления функции ограниченным рядом, а его значение оценивают (как правило, в форме Лагранжа) следующим образом:

$$R_N=\frac{f^{(N+1)}(\delta)}{(N+1)!}(x-a)^{N+1},\quad a<\delta<x$$

При выполнении условия $a=0$, когда все производные вычисляются в нулевой точке, ряд Тейлора приобретает вид, известный как ряд Маклорена:

$$f(x)=\sum_{k=0}^{N}\frac{f^{(k)}(0)}{k!}(x)^k+R_N$$

2. Вывод формулы для ряда Маклорена

Классический вывод коэффициентов степенного ряда в форме ряда Маклорена получается путем аналитического дифференциирования степенного ряда записанного в общем виде.

Пусть является возможным представить некоторую функцию следующим степенным рядом:

$$f(x)=b_0+b_1x^1+b_2x^2+b_3x^3+b_4x^4+\ldots+b_Nx^N+R_N$$

Поскольку степенной ряд достаточно просто дифференцировать, вычислим все его производные до N-го порядка включительно:

$$f^{(1)}(x)=b_1+2\cdot b_2x^1+3\cdot b_3x^2+4\cdot b_4x^3+\ldots+N\cdot b_Nx^{N-1}$$
$$f^{(2)}(x)=2\cdot b_2+3\cdot 2\cdot b_3x^1+4\cdot 3\cdot b_4x^2+\ldots+N\cdot(N-1)\cdot b_Nx^{N-2}$$
$$f^{(3)}(x)=3\cdot 2\cdot b_3+4\cdot 3\cdot 2\cdot b_4x^1+\ldots+N\cdot(N-1)\cdot(N-2)\cdot b_Nx^{N-3}$$
$$f^{(4)}(x)=4\cdot 3\cdot 2\cdot b_4+\ldots+N\cdot(N-1)\cdot(N-2)\cdot(N-3)\cdot b_Nx^{N-4}$$

Далее, применяя индукцию можно записать:

$$f^{(N)}(x) = N \cdot (N-1) \cdot (N-2) \cdot (N-3) \cdot \cdots \cdot (N-(N-1)) \cdot b_N x^{N-N}$$

или

$$f^{(N)}(x) = N! \cdot b_N$$

Вычислим все эти производные в нулевой точке (то есть, при $x = 0$). В этом случае, все члены суммы содержащие x, кроме $x^0 = 1$, обнулятся и вычисленные производные примут вид:

$$f^{(1)}(0) = b_1$$
$$f^{(2)}(0) = 2! b_2$$
$$\cdots$$
$$f^{(N)}(0) = N! \cdot b_N$$

Соответственно коэффициенты b_k, можно легко определить следующим образом:

$$b_k = \frac{f^{(k)}(0)}{k!}; \quad 0 \le k \le N$$

что автоматически позволяет перезаписать исходный степенной ряд в виде ряда Маклорена:

$$f(x) = \sum_{k=0}^{N} \frac{f^{(k)}(0)}{k!}(x)^k + R_N$$

Аналогичным образом можно переопределить степенной ряд и в виде ряда Тейлора.

Как правило, многие функции аппроксимируются именно рядами Маклорена, что несколько упрощает вычисление производных этих функций и позволяет получать более компактные формы записи рядов. Ярким примером служит аппроксимация рядом Маклорена експоненциальной функции:

$$e^x = \sum_{k=0}^{N} \frac{1}{k!}(x)^k + R_N$$

Действительно, если $\frac{d}{dx}(e^x) = e^x$, а любое число в нулевой степени равно единице, то производные всех порядков функции e^x вычисленные в нуле, будут равны единицам и соответствующий ряд примет приведенный выше вид.

Однако, следует также отметить функции, которые для их представления в компактной форме рядом Маклорена, дополнительно требуют вычисления специальных чисел, таких как числа Бернулли или числа Эйлера в форме представления таких чисел для рядов Тейлора.

Так например, тангенс:

$$tg(x) = x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315} + \ldots = \sum_{k=1}^{\infty} \frac{B_{2k}(-4)^k(1-4^k)}{(2k)!} x^{2k-1} + R_N, \quad |x| < \frac{\pi}{2}, \quad B_{2k} - \textit{числа Бернулли}.$$

или секанс:

$$\sec(x) = \sum_{k=1}^{\infty} \frac{E_{2k}(-1)^k}{(2k)!} x^{2k} + R_N, \quad |x| < \frac{\pi}{2}, \quad E_{2k} - \textit{числа Эйлера}.$$

Эти специальные числа мы рассмотрим несколько позже, а пока сосредоточимся на вопросах возможности и точности представления функций с помощью рядов Тейлора и Маклорена.

3. Признаки сходимости рядов Тейлора – Маклорена

Исходное требование, которое формулируется при выводе формул для рядов Тейлора и Маклорена предполагает существование у функции производных любых порядков.

Как правило, это подразумевает, что их значения $f^{(k)}(a)$, являются конечными величинами.

Иначе:

- в случае принципиальной невозможности определения одной или нескольких производных, неопрелеленной будет и вся сумма;
- в случае $x \neq 0$ и бесконечно больших значений (с одним знаком) одной или нескольких производных вся сумма будет бесконечно большой величиной;
- в сучае $x \neq 0$ и знакопеременных бесконечных значений у нескольких производных, вопрос о конечности всей суммы становится нетривиальным и требует отдельного анализа.

Требование конечных значений всех производных, хотя и является необходимым условием, однако не является условием достаточным, поскольку бесконечная сумма конечных величин, также может приводить к бесконечно большим значениям.

Таким образом, вопрос о возможности представления функций с помощью рядов Тейлора и Маклорена, помимо требования существования у функции конечных производных любых порядков, дожен включать и требование об обязательной сходимости этих рядов.

Вопросы сходимости числовых рядов исследовались многими математиками. В результате исследований, было формулировано достаточно много критериев и признаков для анализа сходимости рядов. Наиболее общим является критерий Коши сходимости числового ряда. На основе этого критерия были доказаны признаки сходимости рядов, которые преимущественно используются для проверки сходимости рядов Тейлора и Маклорена, а именно:

а) Признак Даламбера, который формулируется следующим образом:

Если не равные нулю знакоположительные члены p_k некоторого ряда $\sum_{k=0}^{\infty} p_k$ строго удовлетворяют условию $D_{k+1} = \frac{p_{k+1}}{p_k} < 1$ для всех *k*, то данный ряд сходится.

б) Признак Даламбера в предельной форме, который формулируется следующим образом:

Если для не равных нулю знакоположительных членов p_k некоторого ряда $\sum_{k=0}^{\infty} p_k$ строго выполняется $\lim_{k\to\infty} \frac{p_{k+1}}{p_k} = q < 1$, то данный ряд сходится.

в) Признак Даламбера был обобщен Лейбницем для знакочередующихся рядов, то есть:

Если знакочередующиеся члены $(-1)^k p_k$ некоторого ряда $\sum_{k=0}^{\infty} (-1)^k \cdot p_k$ удовлетворяют условию $0 < p_{k+1} < p_k$ и выполняется $\lim_{k\to\infty} (p_k) = 0$, то данный ряд сходится.

Из названных выше признаков, чаще всего, для проверки сходимости рядов, используется признак Даламбера в предельной форме, либо его обобщение, выполненное Лейбницем. Поскольку названные признаки оперируют с знакоположительными членами ряда, то для анализа сходимости ряда удобно применять абсолютные величины этих членов. Рассмотрим несколько примеров:

а) Для функции $f(x) = e^x$, признак обобщенный признак Даламбера использует следующее отношение:

$$|D_{k+1}| = \frac{|p_{k+1}|}{|p_k|} = \frac{\frac{1}{(k+1)!}\cdot|x^k|}{\frac{1}{(k)!}\cdot|x^k|} = \frac{|x|}{(k+1)}$$

для которого достаточно просто найти предел

$$\lim_{k\to\infty}\frac{|p_{k+1}|}{|p_k|} = \lim_{k\to\infty}\frac{|x|}{(k+1)} = |x|\cdot\lim_{k\to\infty}\frac{1}{(k+1)} = |x|\cdot 0 < 1$$

Поскольку умножение на ноль гарантирует результат меньше единицы, то ряд сходится при любых значениях аргумента *x*.

б) Для функции $\ln(x+1) = \sum_{k=1}^{\infty}\frac{(-1)^{k-1}}{k}x^k$ соответствующий признак примет вид:

$$|D_{k+1}| = \frac{|p_{k+1}|}{|p_k|} = \frac{|(-1)^k\frac{x^{k+1}}{k+1}|}{|(-1)^{k-1}\frac{|x^k|}{k}|} = |(-1)|\frac{k}{k+1}|x| = \frac{k}{k+1}|x|$$

$$\lim_{k\to\infty}\frac{|p_{k+1}|}{|p_k|} = \lim_{k\to\infty}\frac{k\cdot|x|}{(k+1)} = |x|\cdot\lim_{k\to\infty}\frac{k}{(k+1)} = |x|\cdot\lim_{k\to\infty}\frac{1}{(1+\frac{1}{k})} = |x| < 1$$

Таким образом этот ряд сходится только на интервале $-1 < x < 1$. При выходе значения *x* за границы этого неравенства ряд начнет расходиться и его значения уже не будут соответствовать значениям аппроксимируемой функции.

4. Оценка остаточных членов рядов Тейлора – Маклорена

Теперь рассмотрим оценки для точности представления некоторой функции рядом Тейлора или Маклорена. Для этого запишем ряд Тейлора в следующем виде:

$$f(x) - \sum_{k=0}^{N}\frac{f^{(k)}(0)}{k!}(x)^k = R_N$$

В этом случае, остаточный член R_N можно рассматривать как ошибку аппроксимации. Для оценки таких ошибок получено несколько оценочных форм, а именно:

- форма Пеано $R_N = o((x-a)^N), \quad при\ x \to a$
- форма Коши $R_N = \frac{f^{(N+1)}(a+\theta_1(x-a))}{N!}(1-\theta_1)^N(x-a)^{N+1}; \quad 0 < \theta_1 < 1$
- форма Лагранжа $R_N = \frac{f^{(N+1)}(\delta)}{(N+1)!}(x-a)^{N+1}, \quad a < \delta < x$

Чаще всего, для оценки остаточного члена в рядах Тейлора или Маклорена используется форма Лагранжа. Запишем эту оценку в виде неравенства, позволяющего оценить максимальную абсолютную ошибку:

$$|R_N| \le \frac{|f^{(N+1)}(\delta)|}{(N+1)!} \cdot |(x-a)^{N+1}|, \quad a < \delta < x$$

В этом случае, необходимо найти такое δ при котором значение производной будет максимальным.

В качестве примера рассмотрим функцию $f(x) = e^x$, при $x = 1$. Выбор аргумента равного единице позволяет рассматривать пример, как задачу об определении длины ряда Тейлора для вычисления значения e (или основания натурального логарифма) с заданной абсолютной ошибкой.
Итак, пусть $a = 0$, тогда соответствующее неравенство в данном примере примет вид:

$$|R_N| \le \frac{|f^{(N+1)}(\delta)|}{(N+1)!}, \quad 0 < \delta < x.$$

Поскольку $f(x) = e^x \rightarrow f^{(k)}(x) = e^x$, или любая производная этой функции является монотонно возрастающей функцией, то ее максимальное значение будет определяться в точке $\delta = x$. В таблице, представленной ниже, для различной длины ряда показаны:

- индекс последнего члена ряда (колонка 1);
- значения факториала для последнего члена ряда (колонка 2);
- соответственно длине ряда вычисленное значение ряда (колонка 3);
- в колонках 4 и 5 показаны значеия остаточных членов для граничных случаев выбора точки δ.

Таблица 1.

N	$N!$	$e^x = \sum_{k=0}^{N} \frac{1}{k!}(x)^k$	$\lvert R_N \rvert;\ \ \delta = x$	$\lvert R_N \rvert;\ \ \delta = 0$
1	2	3	4	5
0	1,00000000000000000E+00	1,00000000000000000	2,718281828	1
1	1,00000000000000000E+00	2,00000000000000000	1,359140914	0,5
2	2,00000000000000000E+00	2,50000000000000000	0,453046971	0,166666667
3	6,00000000000000000E+00	2,66666666666667000	0,113261743	0,041666667
4	2,40000000000000000E+01	2,70833333333333000	0,022652349	0,008333333
5	1,20000000000000000E+02	2,71666666666667000	0,003775391	0,001388889
6	7,20000000000000000E+02	2,71805555555556000	0,000539342	0,000198413
7	5,04000000000000000E+03	2,71825396825397000	6,74177E-05	2,48016E-05
8	4,03200000000000000E+04	2,71827876984127000	7,49086E-06	2,75573E-06
9	3,62880000000000000E+05	2,71828152557319000	7,49086E-07	2,75573E-07
10	3,62880000000000000E+06	2,71828180114638000	6,80987E-08	2,50521E-08
11	3,99168000000000000E+07	2,71828182619849000	5,67489E-09	2,08768E-09
12	4,79001600000000000E+08	2,71828182828617000	4,3653E-10	1,6059E-10
13	6,22702080000000000E+09	2,71828182844676000	3,11807E-11	1,14707E-11
14	8,71782912000000000E+10	2,71828182845823000	2,07871E-12	7,64716E-13
15	1,30767436800000000E+12	2,71828182845899000	1,2992E-13	4,77948E-14
16	2,09227898880000000E+13	2,71828182845904000	7,64233E-15	2,81146E-15
17	3,55687428096000000E+14	**2,71828182845905000**	4,24574E-16	1,56192E-16
18	6,40237370572800000E+15	**2,71828182845905000**	2,2346E-17	8,22064E-18
19	1,21645100408832000E+17	**2,71828182845905000**	1,1173E-18	4,11032E-19
20	2,43290200817664000E+18	**2,71828182845905000**	5,32048E-20	1,95729E-20
21	5,10909421717094000E+19	**2,71828182845905000**	2,4184E-21	8,89679E-22
22	1,12400072777761000E+21	**2,71828182845905000**		

Более точное значение для основания натурального логарифма известно как:

e = 2,7182818284590*45235360287471352*…

5. Вычисление рядов Тейлора – Маклорена в конечной разрядной сетке

Обратим внимание, что начиная с индекса 17 (колонка 1) значения ряда (колонка 2) остаются неизменными и представляют собой округленное до 15 десятичнх знаков значение основания натурального логарифма. Такое округление возникает при суммировании отличных на много порядков действительных чисел в системах программирования с ограниченной разрядной сеткой.

Это достаточно просто пояснить следующим образом:

> Если степень первого знака мантиссы действительного числа в первом операнде суммы равна или больше на величину длины разрядной сетки чем степень первого знака мантиссы второго операнда, то значение второго операнда можно использовать либо в качестве признака округления, либо оно просто исчезает за пределами разрядной сетки.
>
> *Примечание*: сформулированное свойство строго применяется только с учетом конкретной позиционной системы, применяемой для представления действительных чисел. Если используется несколько систем представления чисел (как например, двоичная и десятичная в компьютерах), то оценка делается в той системе, в которой выполняются операции с числами.

В дополнение приведем пример проявления этой особенности в MS EXCEL 2000. При длине десятичной разрядной сетки для мантиссы приблизительно равной 15, мы получим следующие результаты:

```
-1+(-1e-14) = -1,0000000000000100e0
-1+(-1e-15) = -1,0000000000000000e0
```

Поскольку результаты операций деления или вычисления значений многих функций являются действительными числами, данная особенность достаточно часто проявляет себя при программировании систем для инженерных расчетов. Кроме того, некоторые из действительных чисел являются иррациональными. В этом случае, выбор разрядности для действительных чисел неизбежно приводит к ограничению разрядной сетки.

Сегодня представление действительных чисел регулирует «IEEE 754 - стандарт двоичной арифметики с плавающей точкой».

В нашем случае, рассмотренная особенность приводит к следующему выводу:

> Сходящиеся ряды Тейлора-Маклорена при их вычислении в арифметике с плавающей точкой, при достижении некоторой длины ряда *N* могут прекратить ассимтотическое приближение к вычисляемому значению функции. Причиной этому является выход значений очередного ненулевого члена ряда p_{k+1} за пределы значимых величин текущей суммы ряда S_k, если эта сумма вычисляется в конечной разрядной сетке.

Как следствие, в арифметике с плавающей точкой, можно сформулировать признак «предела точности»:

> Если при вычислении некоторого значения непрерывно сходящегося ряда $|D_{k+1}|>0$ условия:
>
> $$|p_{k+1}|>0, \quad S_k - S_{k+1} = 0$$
>
> выполняются один или более раз при увеличении *k*, то это означает, что достигнут предел точности вычисления этого значения в конечной разрядной сетке.

Проверка такого признака в алгоритмах реализуется достаточно просто, однако при этом может существенно ускорить их выполнение.

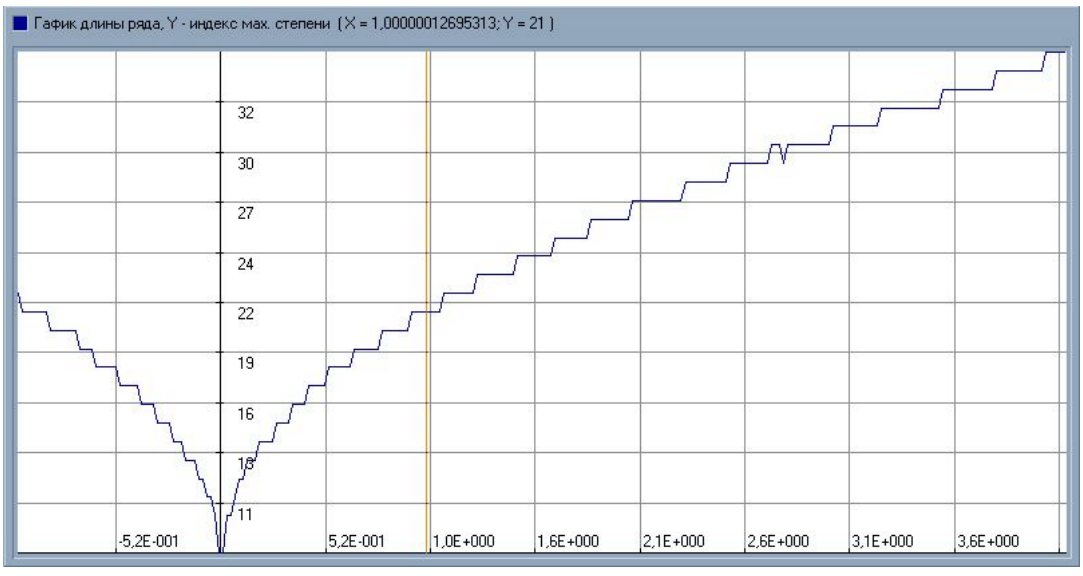
6. Пример программы вычисления ряда Маклорена

В качестве примера, рассмотрим вариант вычисления ряда Маклорена в системе Delphi.

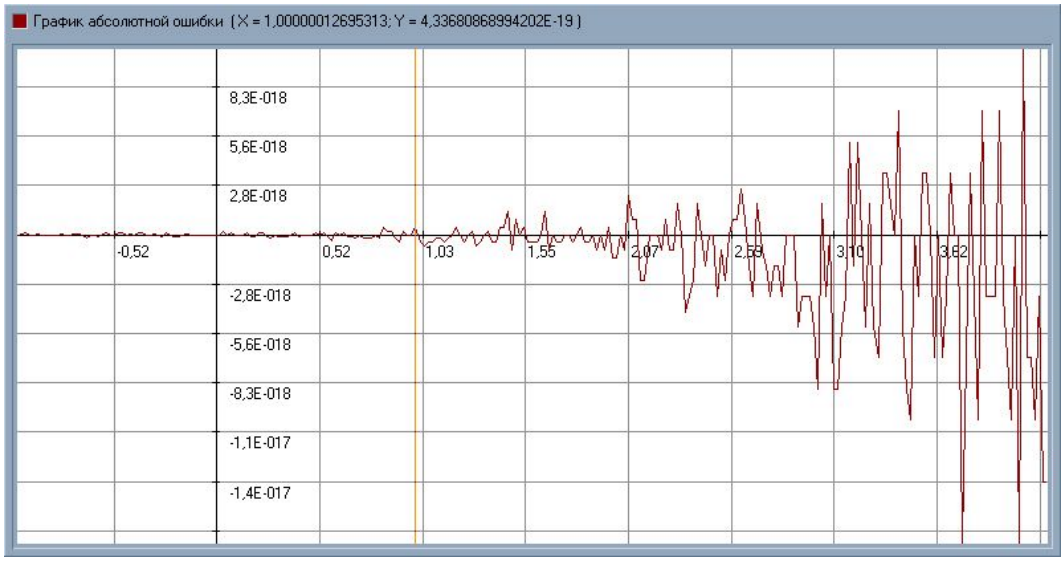
```
// Вычислить сходящийся степенной ряд для конкретного значения X
// Функция возвращает результат вычисления Sum, индекс ряда HInd при котором
// итоговая сумма в конечной разрядной сетеке прекращает изменяться, а также
// и признак безаварийного завернения - Result.
//
function CalcSeriesForX (RqCoeffArr : TD1AttExt;    // Массив коэффициентов ряда
                         RqX        : extended;     // Аргумент
                     var HInd       : integer;      // Итоговый индекс конца ряда
                     var Sum        : extended      // Итоговая сумма ряда
                         ) : boolean;
var Ind       : integer;                            // Рабочий индекс
    PwX       : extended;                           // X в текущей степени
    PSum      : extended;                           // Предшествующее значение суммы
    CtIPL     : integer;                            // Счетчик признака предела точности
begin
   Result := False; PwX := 1; HInd := 0; Sum := 0; CtIPL := 0;
   if Length(RqCoeffArr) < 1 then Exit;
   try
      for Ind := Low(RqCoeffArr) to High(RqCoeffArr)
      do begin
        PSum := Sum; HInd := Ind;
        if (Abs(RqCoeffArr[Ind]) > 0) and (Abs(PwX) > 0)
        then begin                                  // Пропуск нулевых членов ряда
          Sum := Sum + RqCoeffArr[Ind]* PwX;
          if Abs(PSum - Sum) = 0                    // Ловушка признака предела точности
          then begin
              Inc(CtIPL); if CtIPL > 1 then Break;
          end else CtIPL := 0;
        end;
        PwX := PwX * RqX;
      end;
      Result := True;
    except
      Beep;  MessageDlg('Ошибка вычисления элементов для' + #13#10
                      + 'члена ряда с индексом: ' + IntToStr(Ind),
                        mtError, [mbOk], 0);
    end;
end;
```

В системе программирования Delphi Win32, при использовании представления чисел посредством типа extended, рассмотренная особенность суммирования проявляет себя когда различие для первых десятичных знаков операндов достигает значений приблизительно 18 – 19 порядков.

На рисунке, приведенном ниже, показано как изменяется индекс максимальной степени ряда (ось Y) для ряда Маклорена функции $f(x) = e^x$ при достижении предельной абсолютной ошибки для различных значений аргумента функции x (ось X) на участке $-1 < x < 4{,}1717175$.



Как видно из этого численного эксперимента, при вычислении основания натурального алгоритмя в плавающей арифметике Delphi Win32, начиная с длины ряда Маклорена равной 21, значения последующих членов ряда выходят за пределы разрядной сетки суммы, а следовательно алгоритм может быть остановлен. На следующем рисунке показана разница вычисления функции $f(x) = e^x$ рядом Маклорена и стандартной процедурой библиотеки Delphi.



7. Примеры рядов Маклорена

В таблицах 2, … 6, показаны примеры функций представленных рядами Маклорена. При численных экспериментах с этими рядами в системе Delphi Win32 максимальная длина ряда ограничивалась до 256 членов.

В примечаниях к таблицам используются следующие обозначения:

- MaxAE – Максимальная абсолютная ошибка между значениями полученным с помощью функций стандартной библиотеки Math и значениями ряда Маклорена.
- RangAE – Интервал внутри которого выполняется MaxAE
- Ind_IPL – Индекс члена ряда (степень аргумента) в граничных точках интервала RangAE при котором начинает выполняться признак предела точности

Таблица 2.

№	Геометрические ряды и логарифмы	Примечания
1.	$\sqrt{1+x} = \sum_{k=0}^{\infty} \frac{(-1)^k (2k)! x^k}{(1-2k)(k!)^2 (4^k)}; \quad \lvert x \rvert < 1$	MaxAE : | 1e-18 | RangAE : | x | < 0,87 Ind_IPL : 255 (медленная сходимость)
2.	$\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k; \quad \lvert x \rvert < 1$	MaxAE : | 5,6e-18 | RangAE : | x | < 0,85 Ind_IPL : 255 (медленная сходимость)
3.	$\exp(x) = \sum_{k=0}^{\infty} \frac{1}{k!} x^k; \quad x \in C$	MaxAE : | 5,5e-17 | RangAE : - 8 < x < 5 Ind_IPL : 55
4.	$\ln(x+1) = \sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k} x^k; \quad \lvert x \rvert < 1$	MaxAE : | 7,6e-19 | RangAE : | x | < 0,86 Ind_IPL : 255 (медленная сходимость)
5.	$\ln(1-x) = \sum_{k=1}^{\infty} \frac{1}{k} x^k; \quad \lvert x \rvert < 1$	MaxAE : | 7,6e-19 | RangAE : | x | < 0,86 Ind_IPL : 255 (медленная сходимость)

Таблица 3.

№	Тригонометрические функции	Примечания
	$\sin(x)=\sum_{k=0}^{\infty}\frac{(-1)^k}{(2k+1)!}x^{2k+1};\quad x\in C$	MaxAE : | 1e-17 | RangAE : | x | < 6,3 (2* π) Ind_IPL : 63
	$\cos(x)=\sum_{k=0}^{\infty}\frac{(-1)^k}{(2k)!}x^{2k};\quad x\in C$	MaxAE : | 1e-17 | RangAE : | x | < 6,3 (2* π) Ind_IPL : 43
	$\tan(x)=\sum_{k=1}^{\infty}\frac{B_{2k}(-4)^k(1-4^k)}{(2k)!}x^{2k-1};\quad \lvert x\rvert<\frac{\pi}{2}$ B_{2k} - Числа Бернулли для рядов Тейлора	MaxAE : | 1,8e-15 | RangAE : | x | < 1,32 (75,6°) Ind_IPL : 255 (медленная сходимость)
	$\sec(x)=\sum_{k=0}^{\infty}\frac{E_{2k}(-1)^k}{(2k)!}x^{2k};\quad \lvert x\rvert<\frac{\pi}{2}$ E_{2k} - Числа Эйлера для рядов Тейлора	MaxAE : | 5,2e-15 | RangAE : | x | < 1,32 (75,6°) Ind_IPL : 255 (медленная сходимость)

Таблица 4.

№	Обратные тригонометрические функции	Примечания
	$\arcsin(x)=\sum_{k=0}^{\infty}\frac{(2k)!}{4^k(k!)^2(2k+1)}x^{2k+1};\quad \lvert x\rvert\le 1$	MaxAE : | 3,8e-19 | RangAE : | x | < 0,85 Ind_IPL : 255 (медленная сходимость)
	$\arctan(x)=\sum_{k=0}^{\infty}\frac{(-1)^k}{(2k+1)}x^{2k+1};\quad \lvert x\rvert\le 1$	MaxAE : | 3,8e-19 | RangAE : | x | < 0,86 Ind_IPL : 255 (медленная сходимость)

Таблица 5.

№	Гиперболические функции	Примечания
	$\sinh(x)=\sum_{k=0}^{\infty}\frac{1}{(2k+1)!}x^{2k+1};\quad x\in C$	MaxAE : | 8,3e-17 | RangAE : | x | < 6,3 (2* π) Ind_IPL : 45
	$\cosh(x)=\sum_{k=0}^{\infty}\frac{1}{(2k)!}x^{2k};\quad x\in C$	MaxAE : | 4,2e-17 | RangAE : | x | < 6,3 (2* π) Ind_IPL : 44
	$\tanh(x)=\sum_{k=1}^{\infty}\frac{B_{2k}(4^k)(4^k-1)}{(2k)!}x^{2k-1};\quad \lvert x\rvert<\frac{\pi}{2}$ B_{2k} - Числа Бернулли для рядов Тейлора	MaxAE : | 4,3e-15 | RangAE : | x | < 1,32 (75,6°) Ind_IPL : 255 (медленная сходимость)

Таблица 6.

	Обратные гиперболические функции	Примечания
	$ar\sinh(x)=\sum_{k=0}^{\infty}\frac{(-1)^k(2k)!}{4^k(k!)^2(2k+1)}x^{2k+1};\quad \lvert x\rvert<1$	MaxAE : | 3,2e-19 | RangAE : | x | < 0,85 Ind_IPL : 229 (медленная сходимость)
	$ar\tanh(x)=\sum_{k=0}^{\infty}\frac{1}{(2k+1)}x^{2k+1};\quad \lvert x\rvert<1$	MaxAE : | 4,3e-19 | RangAE : | x | < 0,86 Ind_IPL : 255 (медленная сходимость)

Таблица 7.

№	Специальные функции	Примечания
	$\frac{\sin(x)}{x} = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!} x^{2k}; \quad \lvert x \rvert <= 8\pi$	MaxAE : |2e-11 | RangAE : $\lvert x \rvert < 25{,}2 \quad (8^* \pi)$ Ind_IPL : 120
	$si(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!(2k+1)} x^{2k+1}; \quad x \in C$	Интегральный синус
	$Wo(x) = \sum_{k=1}^{\infty} \frac{(-k)^{k-1}}{(2k)!} x^k; \quad \lvert x \rvert < \frac{1}{e}$	Wo- функция Ламберта

8. Способы повышения точности при вычислении функций рядами Тейлора и Маклорена.

Таких способов существует несколько, однако наиболее традиционным из них является использование свойств самоподобия функций и представление функций с разбиением области их опреднления на участки.

8.1. Использование периодического самоподобия функций

Большинство тригонометрических функций повторяют свои значения на любых интервалах кратных их периоду *T*. Такое свойство тригонометрических функций можо использовать для переноса больших значений аргумента функци, где ряды Маклорена начинают заметно отклонятся от соответствующих значений функции, в область ее первого периода. Поскольку производные функции для рядов Маклорена вычисляются при $x = 0$, то в области близкой к нулевым значениям аргумента $|x| \le T$ ряды Маклорена позволяют вычислить значения функции с наилучшей точностью. В этом случае, дополнительные алгоритмические затраты на такой перенос аргумента являются более чем оправданными. Алгоритм этого переноса черезвычайно прост:

Если $|x| > T$, то последовательно выполняя $|x_{k+1}| = |x_k| - T$ добъемся выполнения условия $0 \le |x_{k+1}| \le T$ и, тем самым, получим искомую величину x_T

В тривиальном исполнении алгоритма это простейший цикл. При более внимательном взгляде на алгоритм, легко заметить, что его результатом является остаток от деления величины *x* на величину *T*. Если мы располагаем некоторой функцией *Frac*, которая позволяет получить дробную часть действительного числа (fractional part of a real number), то такой остаток можно получить следующим образом:

$$x_T = T \cdot Frac\left(\frac{x}{T}\right)$$

В большинстве систем программирования функция *Frac* входит в состав библиотеки Math стандартных математических процедур.

Необходимо также заметить, что кроме рассмотренного самоподобия на периоде у отдельных тригонометрических функций (например sin, cos) наблюдается еще ряд дополнительных симметрий. Использование таких симметрий, позволяет заметно сократить необходимую область определения аргументов для вычисления соответствующего ряда Маклорена. Например:

$$\sin(x)\big|_{0 \le x \le \pi} = -\sin(x)\big|_{\pi \le x \le 2\cdot\pi}$$

Как следствие, если для достижения признака предела точности функции sin на границах периода требуется ряд Маклорена у которого степень аргумента в последнем члене равна ~63, то на границах

полупериода необходимая степень снижается до значения ~37, что значительно ускоряет вычисление ряда.

В системе программирования Delphi такое ускорение можно реализовать следующим образом:

```
// SeriesSin(X) - вычисление sin рядом Маклорена
// QuickSeriesSin – быстрое вычисление sin рядом Маклорена
//
function QuickSeriesSin (X : extended) : extended;
begin
  Result := 2 * Pi * Frac(X /(2 * Pi));    // Сместим аргумент в первый период
  if Abs(Result) <= Pi
  then Result := SeriesSin(Result)
  else begin                               // Используем симметрию полупериодов
     Result := Abs(Result) - Pi;           // Сместим аргумент в первый полупериод
     if X >= 0
     then Result := - SeriesSin(Result)
     else Result :=   SeriesSin(Result);
  end;
end;
```

8.2. Использование самоподобия как результата аффинных преобразований

Аффинное преобразование является отображением принципа независимости свойств объекта от выбора точки наблюдения и математически реализуется с помощью операций линейного сдвига точки начала координат, линейного масштатирования, а также векторного вращения. Формально, рассмотренное выше переодическое самоподобие тригонометрических функций можно отнести к операции аффинного сдвига.

В данном подразделе, на примере функции b^x, рассмотрим самоподибие, которое получается в результате операций аффиного сдвига и масштабирования. Итак, для любой показательной функции, включая экспоненту справедливо следующее:

$$f(x) = b^x = b^{a+(x-a)} = b^a \cdot b^{x-a}$$

или

$$f(x) = b^x = b^a \cdot b^\delta, \quad \delta = x - a,$$

где точку (a) можно рассматривать как точку нового начала координат независимой переменной δ, а множитель b^a как масштабный множитель.Однако, важно отметить следующее:

если $a = 0$ то $b^x = b^\delta$

Следовательно, показательную функцию можно представить как периодическую, если на каждом периоде (участке) с индексом k ее аргумент будет изменяться в пределах интервала $0 \le \delta \le a$, а для получения конечного результата b^x на конкретном периоде функции применить аффинные операции, то есть:

$$0 \le \delta = x - a_k \le a, \quad a_k = k \cdot a, \quad k \in (0 \ldots K) \text{ (операция сдвига)}$$

$$b^x = b^{a_k} \cdot b^\delta \qquad \text{(операция масштабирования)}$$

Другими словами, нами показано самоподобие потенциальной функции на различных участках ее аргумента, достигнутое в следствии применения аффинных преобразований к значениям функции на ее начальном участке k=0. С точки зрения простейшего алгоритма, который реализует вычисление функции с применением участков, это означает необходимость использование массива предварительно вычисленных констант для операций масштабирования на этих участках (периодах).

Наличие такого массива или иного способа получить масштабирующие константы b^{a_k} каждого участка, является естественной ценою за сокращенный диапазон аргументов в котором вычисляется названная функция и может обосновываться повышением скорости и точности вычисления, особенно при вычислении функций сходящимися рядами.

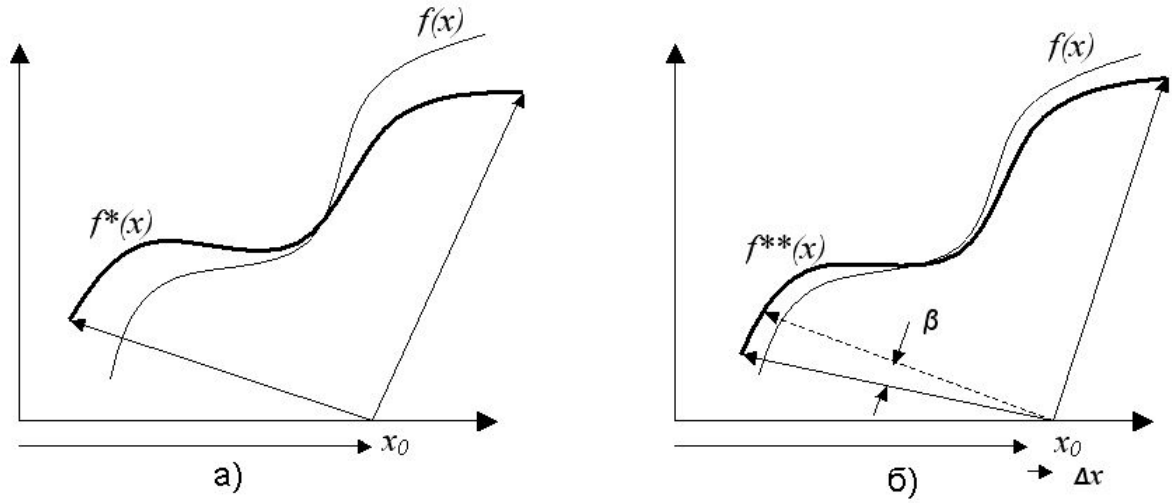
С позиции применения операций сдвига и масштабирования достаточно любопытно посмотреть на ряд Тейлора:

$$f(x) = \sum_{k=0}^{N} \frac{f^{(k)}(a)}{k!}(x-a)^k + R_N$$

Легко заметить, что этот ряд можно интепретировать как сумму функций x^k, начало координат которых сдвинуто в точку $x = a$, а в качестве масштабов используются константы $f^{(k)}(a)/k!$. Такой взгляд позволяет еще раз подчеркнуть фундаментальность свойств аффинного преобразования.

Как уже упоминалось ранее, в состав операций аффинного преобразования входит также операция векторного вращения точек некоторого объекта, относительно произвольно выбранной точки, именуемой центром вращения. Поскольку вопросы применения этой операции существенно уводят нас от рассмотрения рядов Тейлора и Маклорена для действительного аргумента, ограничемся лишь иллюстрацией такого подхода в графическом виде.

Итак, если на некотором участке функция *f(x)* и ее аппроксимация рядом *f*(x)* обладают сходной кривизной, как это показано на рисунке (а), то вращение на угол β каждой точки функции *f*(x)* относительно центра вращения x_0 и последующий сдвиг $x_0 + \Delta x$ (для возврата точек в область участка), позволяет заметно уменьшить абсолютную ошибку аппроксимации. На рисунке (б) показана функция *f**(x)* как конечный результат применения операций вращения и последующего сдвига.



В инженерной практике полный набор аффинных операций обычно применяется в тех случаях, когда располагая рядом Маклорена с уже вычисленными коэффициентами, по разным причинам затруднительно или невозможно вычислить значения производных в точках отличных от нуля. Однако в случае когда производные всех порядков могут быть вычислены в любой точке рассматриваемой области определения, как правило, для повышения точности область определения функции разбивается на участки и для каждого участка вычисляется соответствующий ряд Тейлора. Возможность представлять функции отдельными рядами на участках выбранной длины позволяет не только повысить точность аппроксимации таких функций, но и повысить скорость алгоритмов за счет сокращения длины рядов. При этом следует помнить, что приходится тем или иным способом увеличивать число хранимых или вычисляемых значений, которые описывают параметры рядов на этих участков.

9. Специальные числа для рядов Тейлора – Маклорена

9.1 Числа Бернулли

История специальных чисел во многом обусловлена задачей вычисления суммы последовательных натуральных чисел, возведённых в одну и ту же степень:

$$S_{K-1} = \sum_{k=1}^{K-1} k^n$$

Такая сумма была вычислена Якобом Бернулли в следующей форме:

$$S_{K-1} = \sum_{k=1}^{K-1} k^n = \frac{1}{n+1} \cdot \sum_{i=0}^{n} C_{n+1}^{i} \cdot B_i \cdot K^{n+1-i}$$

Приведенная сумма записана в весьма компактном виде, что оказалось возможным благодаря ряду специальных чисел, а именно - биномиальных коэффициентов C_n^k, а также чисел Бернулли B_k. Эти специальные числа определяются следующим образом:

Биномиальные коэффициенты – это последовательность рациональных чисел, например: $C_0^K, C_1^K, C_2^K, \cdots, C_K^K$, которая вычисляется с помощью факториалов:

$$C_n^k = \frac{n!}{k!(n-k)!}$$

Числа Бернулли — это последовательность рациональных чисел $B_0, B_1, B_2, \cdots, B_K$, которая может вычисляться рекуррентной формулой:

$$B_0 = 1, \quad B_k = \frac{-1}{k+1} \cdot \sum_{n=1}^{k} C_{n+1}^{k+1} \cdot B_{k-n}$$

Кроме того, все числа Бернулли с нечетным индексом приравниваются нулю.

В некоторых случаях производится переиндексация последовательности для того, чтобы опустить нечетные номера чисел Бернулли с нулевым значением (B_{2n}). Обычно такие числа Бернулли применяются в при разложениях в ряд Тейлора функций *tan(x)* и *tanh(x)*.

Таблица 8. Биномиальные коэффициенты и числа Бернулли, вычисленные в конечной разрядной сетке.

БИНОМИАЛЬНЫЕ КОЭФФИЦИЕНТЫ для фиксированного K= 64		ЧИСЛА БЕРНУЛЛИ для максимального K= 64	
n = 0	1	k = 0	1
n = 1	64	k = 1	-0,5
n = 2	2016	k = 2	0,166666666666667
n = 3	41664	k = 3	0
n = 4	635376	k = 4	-0,0333333333333333
n = 5	7624512	k = 5	0
n = 6	74974368	k = 6	0,0238095238095238
n = 7	621216192	k = 7	0
n = 8	4426165368	k = 8	-0,0333333333333333
n = 9	27540584512	k = 9	0
n = 10	151473214816	k = 10	0,0757575757575758
n = 11	743595781824	k = 11	0
n = 12	3284214703056	k = 12	-0,253113553113553
n = 13	13136858812224	k = 13	0
n = 14	47855699958816	k = 14	1,16666666666667
n = 15	159518999862720	k = 15	0
n = 16	488526937079580	k = 16	-7,0921568627451
n = 17	1,37937017528352E15	k = 17	0
n = 18	3,60168879101808E15	k = 18	54,9711779448622
n = 19	8,71987812562272E15	k = 19	0
n = 20	1,96197257826511E16	k = 20	-529,124242424242
n = 21	4,11079968779357E16	k = 21	0
n = 22	8,03474484432379E16	k = 22	6192,1231884058
n = 23	1,46721427592E17	k = 23	0
n = 24	2,50649105469666E17	k = 24	-86580,2531135531
n = 25	4,01038568751466E17	k = 25	0
n = 26	6,01557853127199E17	k = 26	1425517,16666667
n = 27	8,46636978475317E17	k = 27	0
n = 28	1,11877029298524E18	k = 28	-27298231,0678161
n = 29	1,3888182947403E18	k = 29	0
n = 30	1,62028801053035E18	k = 30	601580873,900642
n = 31	1,77709007606554E18	k = 31	0
n = 32	1,83262414094259E18	k = 32	-15116315767,0922
n = 33	1,77709007606554E18	k = 33	0
n = 34	1,62028801053035E18	k = 34	429614643061,167
n = 35	1,3888182947403E18	k = 35	0
n = 36	1,11877029298524E18	k = 36	-13711655205088,3
n = 37	8,46636978475317E17	k = 37	0
n = 38	6,01557853127199E17	k = 38	488332318973593
n = 39	4,01038568751466E17	k = 39	0
n = 40	2,50649105469666E17	k = 40	-1,92965793419401E16
n = 41	1,46721427592E17	k = 41	0
n = 42	8,03474484432379E16	k = 42	8,41693047573683E17
n = 43	4,11079968779357E16	k = 43	0
n = 44	1,96197257826511E16	k = 44	-4,03380718540595E19
n = 45	8,71987812562272E15	k = 45	0
n = 46	3,60168879101808E15	k = 46	2,1150748638082E21
n = 47	1,37937017528352E15	k = 47	0
n = 48	488526937079580	k = 48	-1,20866265222965E23
n = 49	159518999862720	k = 49	0
n = 50	47855699958816	k = 50	7,50086674607696E24
n = 51	13136858812224	k = 51	0
n = 52	3284214703056	k = 52	-5,03877810148107E26
n = 53	743595781824	k = 53	0
n = 54	151473214816	k = 54	3,65287764848181E28
n = 55	27540584512	k = 55	0
n = 56	4426165368	k = 56	-2,84987693024509E30
n = 57	621216192	k = 57	0
n = 58	74974368	k = 58	2,38654274996836E32
n = 59	7624512	k = 59	0
n = 60	635376	k = 60	-2,13999492572253E34
n = 61	41664	k = 61	0
n = 62	2016	k = 62	2,05009757234781E36
n = 63	64	k = 63	0
n = 64	1	k = 64	-2,09380059113464E38

9.2. Числа Эйлера для рядов Тейлора

В теории чисел, числа Эйлера это последовательность *En* целых чисел, которые определяеются в результате разложение в ряд Тейлора выражения:

$$\frac{1}{\cosh(t)} = \frac{2}{e^t + e^{-t}} = \sum_{n=0}^{\infty} \frac{E_n}{n!} \cdot t^n$$

где *cosh(t)* гиперболический косинус. В общем случае, числа Эйлера являются специальными значениями многочленов Эйлера. Все нечетные числа Эйлера равны нулю. В некоторых случаях производится переиндексация последовательности для того, чтобы опустить нечетные номера чисел Эйлера с нулевым значением (E_{2n}). Обычно такие числа Эйлера применяются в при разложениях в ряд Тейлора функций *sec(x)* и *sech(x)*. Они также получаются в комбинаторике, в частности, при подсчете количества чередующихся перестановок на множестве с четным числом элементов.

Явная формула для чисел Эйлера:

$$E_{2n} = i \sum_{k=1}^{2n+1} \sum_{j=0}^{k} \binom{k}{j} \frac{(-1)^j (k-2j)^{2n+1}}{2^k i^k k}$$

где ***i*** обозначает мнимую единицу $i^2 = -1$.

Таблица 9. Некоторые числа Эйлера, вычисленные алгоритмами с неограниченной разрядной сеткой

N	En
0	1
1	0
2	-1
3	0
4	5
5	0
6	-61
7	0
8	1385
9	0
10	-50521
11	0
12	2702765
13	0
14	-199360981
15	0
16	19391512145
17	0
18	-2404879675441
19	0
20	370371188237525
21	0
22	-69348874393137901
23	0
24	15514534163557086905
25	0
26	-4087072509293123892361
27	0
28	1252259641403629865468285

В программных приложениях, необходимость вычисления чисел Эйлера достаточно часто возникает совместно с необходимостью вычислять числа Бернулли. В этом случае для вычисления чисел Эйлера обычно используется формула следующего вида:

$$E_k = \sum_{n=1}^{k} C_{n-1}^{k} \frac{(2^n - 4^n)}{n} B_n, \quad E_0 = 1, E_1 = 0, k \geq 2.$$

где C_{n-1}^{k} соответствующие биномиальные коэффициенты, а *Bn* – соответствующие числа Бернулли.

9.3. Пример программ для вычисления чисел Бернулли и Эйлера

Числа Бернулли и Эйлера достаточно вычисляются, например в Delphi:

```
// ------------------------------------------------------------------------
// Тип для одномерного динамического массива значений специальных чисел
type TD1AttExt = array of extended;
// ------------------------------------------------------------------------
// Факториал
function Fact(RqInd : integer) : extended;
var Ind  : word;
begin
    Result := 1;
    if RqInd <= 0 then Exit;
    for Ind := 1 to RqInd do Result := Result * Ind;
end;

// ------------------------------------------------------------------------
// Биномиальный коэффициент
function Ckn(k,              // Верхний индекс
             n : integer     // Нижний индекс
             ) : extended;
begin
```

```
   Result := Fact(k) / Fact(n);
   Result :=  Result / Fact(k-n);
end;
// ---------------------------------------------------------------------------
// Построить массив биномиальных коэффициентов с изеняющимся индексом - n
// при заданном верхнем индесе (RqK)
procedure MakeCknArr(RqK : integer; var Arr : TD1AttExt);
var n : integer;
begin
    SetLength(Arr, RqK + 1);
    for n := 0 to RqK do Arr[n] := Ckn(RqK, n);
end;
// ---------------------------------------------------------------------------
// Построить массив Arr с числами Бернулли с максимальным индексом RqK
procedure MakeBernoulliArr(RqK : integer; var Arr : TD1AttExt);
var n, k, i    : integer;
begin
   SetLength(Arr, RqK + 1);
   Arr[0] := 1;
   for k := 1 to RqK
   do begin
      Arr[k] := 0;
      for n := 1 to k
      do Arr[k] := Arr[k] + (-1 /(k + 1)) * Ckn(k + 1, n + 1)* Arr[k - n];
   end;
   for i := 2 to RqK
   do if (i mod 2) <> 0 then Arr[i] := 0;
end;
// ---------------------------------------------------------------------------
// Построить динамический массив Arr с числами Эйлера в количестве RqK
procedure MakeTEulerArr(RqK : integer; var Arr : TD1AttExt);
var k, n  : integer;
    Item  : extended;
    BArr  : TD1AttExt;
begin
   // Построить массив Arr с числами Бернулли
   MakeBernoulliArr(RqK, BArr);
   // Подготовить массив чисел Эйлера
   SetLength(Arr, RqK + 1);
   Arr[0] := 1; Arr[1] := 0;
   for k := 2 to RqK
   do begin
     Arr[k] := 0;
     if BArr[k] <> 0
     then begin
        for n := 1 to k
        do begin
          Item := Ckn(k, n-1) * BArr[n] * (IntPower(2, n) - IntPower(4, n)) /n;
          Arr[k] := Arr[k] + Item;
        end;
     end;
   end;
end;
// ---------------------------------------------------------------------------
```

Для ясного восприятия, исходные тексты программ приведены в максимально упрощенном виде. Из теста искючены элементы входного контроля, элементы перехвата исключительных ситуаций и т.д. Но даже в таком виде процедуры безаварийно позволяют строить таблицы до 1024 чисел.

Например для чисел Бернулли:

```
......
k = 1019        0
k = 1020        -7,1251404225847E1813
k = 1021        0
k = 1022        1,88326020311677E1818
k = 1023        0
k = 1024        -4,99719368710874E1822
```

Означает ли это высокую точность вычисления специальных чисел? Для ответа на данный вопрос необходимо еще раз подчеркнуть, что во многих языках количество символов (десятичных знаков) для представления действительных чисел в бинарной форме ограничена.

Ранее (при рассмотрении сходящихся рядов) мы уже отмечали, что для члена ряда степень первой значащей цифры которого была меньше степени последней значащей цифры в текущей сумме, наблюдалась потеря значения такого члена ряда в итоговой сумме. При вычислении специальных чисел имеет место зеркальная картина. Другими словами, если член суммы такой, что степень его последней цифры больше, чем степень последней цифры в сумме (представленной максимальным количеством знаков), то происходит вытеснение последней цифры итоговой суммы за пределы

разрядной сетки. Например, рассмотрим результат вычисления чисел Бернулли алгоритмами с неограниченной разрядной сеткой (таблица 8). Легко заметить, что начиная где-то с B(36) или B(40) числитель дроби (в случае Delphi) должен автоматически округляться в среднем до 18 значащих цифр.

Таблица 8. Числа Бернулли, вычисленные алгоритмами с неограниченной разрядной сеткой

```
B(0 ) =                                                1 / 1
B(1 ) =                                               -1 / 2
B(2 ) =                                                1 / 6
B(4 ) =                                               -1 / 30
B(6 ) =                                                1 / 42
B(8 ) =                                               -1 / 30
B(10) =                                                5 / 66
B(12) =                                             -691 / 2730
B(14) =                                                7 / 6
B(16) =                                            -3617 / 510
B(18) =                                            43867 / 798
B(20) =                                          -174611 / 330
B(22) =                                           854513 / 138
B(24) =                                       -236364091 / 2730
B(26) =                                          8553103 / 6
B(28) =                                    -23749461029 / 870
B(30) =                                   8615841276005 / 14322
B(32) =                                 -7709321041217 / 510
B(34) =                                  2577687858367 / 6
B(36) =                       -26315271553053477373 / 1919190
B(38) =                            2929993913841559 / 6
B(40) =                      -261082718496449122051 / 13530
B(42) =                      1520097643918070802691 / 1806
B(44) =                  -27833269579301024235023 / 690
B(46) =                  5964511115939121632779 61 / 282
B(48) =              -560940336899781768624912 7547 / 46410
B(50) =                49505720524107964821247 7525 / 66
```

Как следствие, попытки неограничено увеличивать длину ряда Маклорена для повышения точности аппроксимациив системах программирования с конечной разрядной сеткой могут привести к прямо противоположному результату. Например, для функции *tan(x)*, в которой применяются числа Бернулли, малая скорость сходимости требует увеличения длины ряда, а потеря точности коэффициентов (при конечной разрядной сетки) требует уменьшение длины этого ряда. Поиск оптимального баланса между этими требованиями безусловно интересен с научной точки зрения, однако далеко выходит за пределы инженерных задач.

Таким образом, как вывод, можно сформулировать потребность в поиске альтернативных методов для аппроксимации функций с медленной сходимостью при их представлении рядами Тейлора или Маклорена.

10. Краткий обзор некоторых альтернативных аппроксимаций для различных функций

Использование степенных рядов, как основной формы для аппроксимации функций, обусловлено тем, что степенной ряд описывается только с помощью элементарных арифметических операций: суммирование, вычитание, умножение и деление. Однако на базе этих операций можно построить и другие регулярные или итерационные структуры.

Рассмотрим некоторые, наиболее популярные структуры регулярного или итерационного типа.

10.1. Функции как отношение двух степеных рядов

Отношение двух степеных рядов обычно записывается в следующей форме:

$$f(x) = \frac{\sum_{n=0}^{\infty} a_n x^n}{\sum_{m=0}^{\infty} a_m x^m}$$

Хорошим примером такого отношения является отношение рядов представляющих соответственно функции $\sin(x)$ и $\cos(x)$, что прямым образом применяется для вычисления tan(*x*) в библиотеке Math системы Delphi.

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

Ряды $\sin(x)$ и $\cos(x)$ достаточно хорошо сходятся и, при относительно небольшой длине (~60-70) членов, обеспечивают высокую точность. Это позволяет с высокой точностью получить функции $\tan(x)$ и $\operatorname{cotan}(x)$ в широком диапазоне их аргумента. Напомним, что для функций $\tan(x)$ и $\operatorname{cotan}(x)$ обычный ряд Маклорена сходится очень медленно, а в области близкой к точкам разрыва теоретически требует бесконечного числа членов, что при конечной разрядной сетке приводит к серьезным проблемам.

Также полезно отметить, что если удается найти действительные или комплексные корни степенных рядов (x_n и x_m), то согласно формулам Виета отношение двух степеных рядов можно представить в следующем виде:

$$f(x) = \frac{\sum_{n=0}^{N} a_n x^n}{\sum_{m=0}^{M} a_m x^m} = \frac{\prod_{n=0}^{N} (x - x_n)}{\prod_{m=0}^{M} (x - x_m)}$$

В инженерной практике такая форма используется для представления различных передаточных функций.

10.2. Вычисление функций посредством цепных дробей

Одной и той же функции можно сопоставить различные модели, выбор которых зависит от решаемой задачи. Для широкого класса функций с точки зрения возможности получения их значений с наперед заданной точностью за наименьшее количество арифметических действий (за наименьшее машинное время) наилучшими моделями являются подходящие дроби цепных дробей.

Цепной (непрерывной) дробью [7], называется выражение вида:

$$b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \dots}}$$

Из-за громоздкости записи цепная дробь обычно записывается в следующих формах:

$$b_0 + \frac{a_1}{b_1 +} \quad \frac{a_2}{b_2 +} \quad \dots \quad \frac{a_n}{b_n} \quad \dots, \quad b_0 + K_{n=1}^{\infty}\left(\frac{a_n}{b_n}\right), \quad b_0 + \Phi\left(\frac{a_n}{b_n}\right),$$

где $\frac{a_n}{b_n}$ называют *n*-звеном цепной дроби.

В [7] доказано: для комплексных $z \neq (2n+1)(\pi/2)$ где *n*-целое, справедливо разложение в цепную дробь:

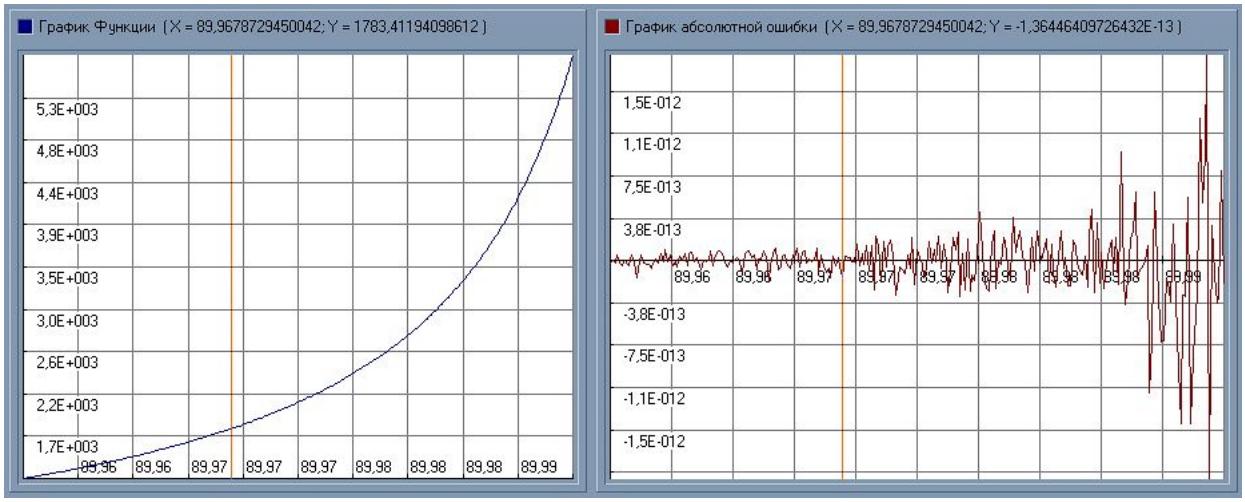
$$\tan(z) = \frac{z}{1-} \quad \frac{z^2}{3-} \quad \dots \quad \frac{z^2}{2n-1-} \quad \dots$$

Для действительного аргумента реализация в Delphi такой цепной дроби будет иметь вид:

```
// Итерационный алгоритм вычисления тангенса цепной дробью
function tgx(X : extended) : extended;
var n, i : integer;
```

```
    s, g : extended;
begin
     n := 16;                          // Общее число звеньев дроби
     g := 0;
     Result := g;
     if X = 0 then Exit;
     s := X * X  / (2 * n -1);         // Последнее звено цепи
     for i := n -1 downto 1
     do begin
         g := X * X / (2 * i - 1 - s); // Очередное звено цепи
         s := g;
     end;
     Result := g / X;                  // Поправить степень первого звена
end;
```

Как видно из исходного текста, длина дроби составляет всего 16 звеньев. При этом в диапазоне от 0 до 89,990 градуса абсолютная ошибка аппроксимации функции tan(x) не хуже чем 2e-12. На рисунке представленом ниже, показан график tan(x) в близи точки разрыва (90 градусов). Абсолютная ошибка вычислялась относительно библиотечной функции Math.Tan(x).



В рамках этого примера полезным также отметить, что по значениям tan(x) можно вычислить sin(x) и cos(x) используя формулы:

$$\sin(x) = \frac{2\tan(x/2)}{1+\tan^2(x/2)}; \quad \cos(x) = \frac{1-\tan^2(x/2)}{1+\tan^2(x/2)}$$

Вычисление tan(x) с помощью цепной дроби, при сходных характеристиках точности, заметно быстрее чем метод, рассмотренный в 10.1. Такая особенность позволяет рекомендовать данный метод для систем, которые работают в режиме реального времени.

10.3. Вычисление функций посредством геометрических преобразований

Вычисление функций посредством геометрических преобразований является одним из разновидностей итерационных методов. Особенностью этого метода является преобразование числовых характеристик геометрических фигур для вычисления соответствующих функций, которыми связаны эти числовые характеристики.
В качестве примера, рассмотрим вычисления корней различной степени от действительного, положительного аргумента.

Для начала сформулируем логику метода для вычисления квдратного корня.

> *Квадратный корень*. Возьмем прямоугольник со сторонами a=1 и b=X. Площадь этого прямоугольника равна S = a * b = 1*X = X. Преобразовав прямоугольник в квадрат так, что его площадь останется прежней, получим длину стороны , равную корню квадратному из площади фигуры, то есть из значения X.

Преобразование прямоугольника в квадрат будем выполнять по следующим итерационным формулам:

$$a_{n+1} = (a_n + b_n)/2, \quad b_{n+1} = S/a_{n+1}$$

При этом, каждые следующие значения *a* и *b* будут усредняться, приближаясь к равенству *a* = *b*, однако площадь фигуры будет оставаться неизменной. Логика этого метода программым кодом реализуется следующим образом:

```
// Итерационный алгоритм вычисления корня квадратного методом
// преобразования прямоугольника в квадрат
function sqrt_ab(X : extended) : extended;
const EPS = 1e-18;           // Необходимая относительная ошибка
var s, a, b : extended;
begin
  Result := 0;
  if X < EPS then Exit;
  s := Abs(X); a := 1; b := s;
  while Abs(1 - b/a) > EPS
  do begin
    a := (a+b)/2;            // Очередное среднее значение сторон
    b := s/a;                // Значение b для сохранения неизменной площади
  end;
  Result := (a+b)/2;
end;
```

Число итераций (для достижения абсолютной ошибки 1e-18) в среднем диапазоне аргумента X (от 0,9999e-14 до 0,9999e14) составляет приблизительно 25 - 30 итераций. Таким образом, мы получаем высокоточный и быстрый алгоритм, пригодный для применения в системах реального времени.

Логику метода вычисления квадратного корня достаточно просто распространить на методы вычисления кубического корня и корня произвольной целой степени. В этом случае логику можно сформулировать следующим образом:

Корень кубический. Возьмем прямоугольную пирамиду со сторонами a=1, a=1 и b=X. Объем этой пирамиды равен V = a * a * b = 1*1*X = X. Преобразовав пирамиду в куб так, что его объем останется прежним, получим длину стороны куба, равную корню кубическому из объема фигуры.

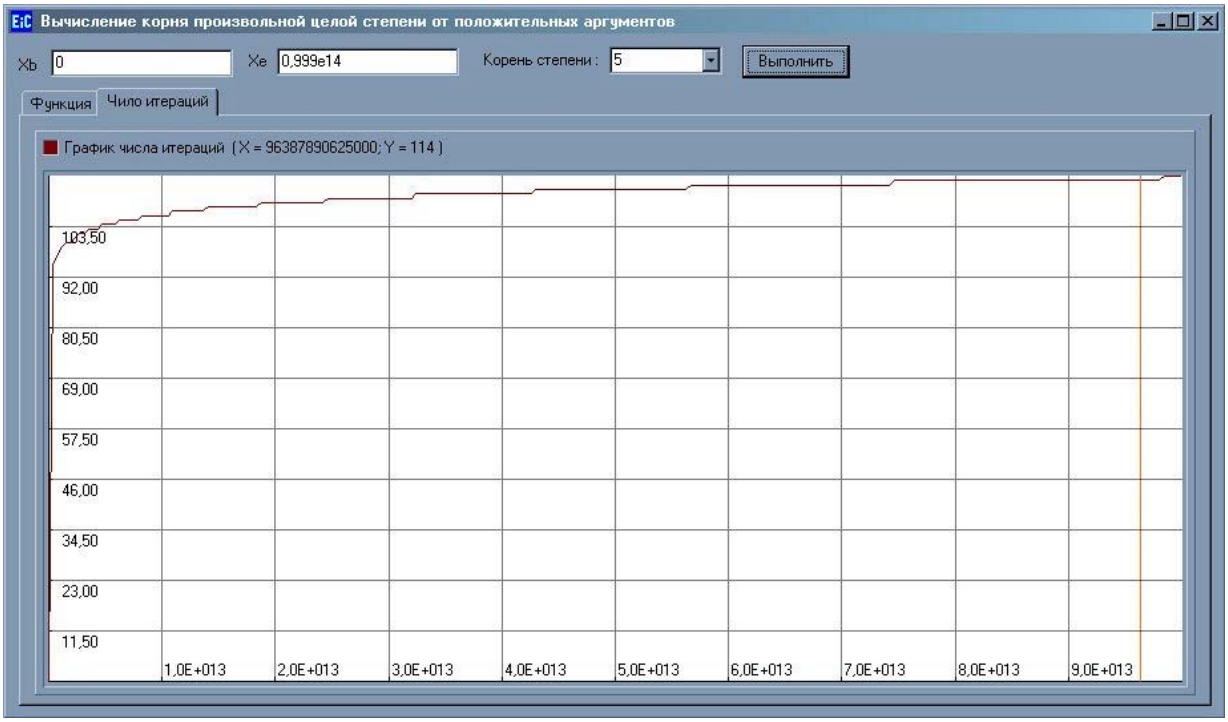
Корень N-степени. Возьмем прямоугольную гиперпирамиду с (N-1) ребрами a=1 и ребром b=X. Гиперобъем этой пирамиды равен V = (N-1) * a * b = 1 * X = X. Преобразовав пирамиду в гиперкуб так, что его гиперобъем останется прежним, получим длину ребра гиперкуба равную корню N-степени из гиперобъема этой фигуры.

Соответствующие методам исходные тексты имеют следующий вид:

```
// ------------------------------------------------------------
// Итерационный алгоритм вычисления кубического корня
function Root3_ab(X : extended; var Count : integer) : extended;
const EPS = 1e-18;           // Необходимая относительная ошибка
var a, b : extended;
begin
  Count := 0;
  Result := 0;
  if Abs(X) < 0 then Exit;
  a := 1; b := Abs(X);
  while (Abs(1-b/a) > EPS)
  do begin
    a := (2*a + b)/3;        // Очередное среднее значение ребер
    b := X/a; b := b/a;      // Значение b для сохранения неизменного объема
    Inc(Count);
  end;
  Result := (2*a + b)/3;
end;
// ------------------------------------------------------------
// Итерационный алгоритм вычисления корня степени N
function RootN_ab(X : extended; N : word; var Count : integer) : extended;
const EPS = 1e-18;           // Необходимая относительная ошибка
var a, b  : extended;
      i   : integer;
begin
```

```
  Count  := 0;
  Result := 0;
  if (Abs(X) = 0) or (N < 2) then Exit;
  a := 1; b := Abs(X);
  while Abs(1 - b/a) > EPS
  do begin
    a := ((N-1)*a + b)/N;    // Очередное среднее значение ребер
    // Значение b для сохранения неизменного гиперобъема
    b := Abs(X);
    for i:=1 to (N-1) do b := b/a;
    Inc(Count);
  end;
  Result := ((N-1)*a + b)/N;
end;
```

Параметр функций Count используется для получения количества итераций, при которых достигается заданная абсолютная ошибка EPS. Знак аргумента X игнорируется, то есть, корни в любом случае вычисляются для положительных значений X. На рисунке, который представлен ниже, показана динамика количества итераций для корня пятой степени в диапазоне аргументов от 0 до 0,999e14 для достижения относительной ошибки 1e-18.



Примечание. В алгоритмах вычисления корней вместо оценки точности в виде абсолютной ошибки применяется оценка по относительной ошибке. Это обусловлено широким диапазоном аргумента для которого условие выхода из цикла по абсолютной ошибке при больших значениях X оказывается недостижимым в силу конечной разрядной сетки.

Список литературы, ссылки

1. ***Корн Г., Корн Т.*** Справочник по математике (для научных работников и инженеров). М.: Наука, - 1978, - 832с.: ил.
2. ***Тиман А.Ф.*** Теория приближения функций действительного переменного. М.: Государственное издательство физико-математической литературы, -1960, - 624с.: ил.
3. ***Ильин В. А., Садовничий В. А., Сендов Б. Х.*** Математический анализ, ч. 1, изд. 3, ред. А. Н. Тихонов. М.: Проспект, 2004.
4. ***Hazewinkel, Michiel,*** ed. (2001), "Euler numbers", Encyclopedia of Mathematics, Springer, ISBN 978-1-55608-010-4
5. ***Abramowitz, M.; Stegun, C. A.*** (1972), "§23.1: Bernoulli and Euler Polynomials and the Euler-Maclaurin Formula", Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables (9th printing ed.), New York: Dover, pp. 804–806.
6. ***Библиография*** (специальные числа) по ссылке: https://en.wikipedia.org/wiki/Bernoulli_number

7. ***Wall, H. S.*** Analytic Theory of Continued Fractions. New York: Chelsea, 1948.
8. ***Olds, C. D.*** Continued Fractions. New York: Random House, 1963.
9. ***Джоунс У., Трон У.*** Непрерывные дроби, Аналитическая теория и положения. – М.: Мир, 1985. – 414 с.
10. ***Хованский А.Н.*** Приложение цепных дробей и их обобщений к вопросам приближенного анализа. – М.:ГИИТТЛ, 1956. – 203 с.