Шнейвайс А.Б.

Лабораторные работы по программированию.

Приведение формул к виду

удобному для расчета

Первая часть:

ФОРТРАН (g77),

СИ (gcc)

2008

Содержание

1 Введение.

Практика по курсу "Программирование и математическое обеспечение ЭВМ", выполняемая первокурсниками астрономического отделения для закрепления тем, пройденных в первые два месяца обучения, проводится на примере решения задачи табулирования некоторой простой функции и состоит из ряда лабораторных работ.

В целом работы нацелены на выработку у обучающегося некоторых элементарных навыков программирования, которые позволят ему эффективно использовать их при проведении курсовых работ расчетного характера.

В то же время выполнение каждой лабораторной преследует определенную цель. Например, лабораторные работы **0a**, **0b** и **0c** должны прояснить, что далеко не всегда синтаксически верное (формальное) программирование *в лоб* формулы, приведенной в условии, способно дать правильный числовой результат. Вопросы решаемые при выполнении каждой лабораторной указаны в приведенных ниже таблицах

Номер ЛР	**Назначение лабораторной работы**
0a	**Убедиться, что численно результат работы исполнимого файла, полученного из исходных текстов, приведенных в условии, практически не отличается и от приведенного результата.**
0b	Перевести исходные тексты в режим работы на удвоенной точности. **Убедиться, что получаемый результат существенно отличается от результата одинарной.**
0c	**Письменно:** 1) оценить правильность обоих результатов, 2) аналитически найти предельное значение табулируемой функции 3) указать объективную и субъективную причины неудовлетворительной работы алгоритма расчета в режиме одинарной точности. 4) аналитически вывести формулу, которая и в режиме одинарной точности позволяет получать высокоточный результат (6-7 верных десятичных значащих цифр).
1a	Модификация исходных ФОРТРАН-текстов (с учетом выполнения **0c**), которая позволяет вычислять искомую функцию обоими способами: **старым** и **новым**. Реализация пропуска и обзора результатов работы программы (в виде таблицы и графиков) посредством вызовов элементарных **make**-файла и **gnuplot**-скрипта при создании в текущей директории файла с рисунком, соответствующим **eps**-формату. **Убедиться, что результаты пропуска (таблица и график) соответствует ожидаемому повышению точности.**
1b	Реализация ФОРТРАН-алгоритма в режиме удвоенной точности Убедиться в ожидаемом повышении точности. Найти область изменения аргумента, на которой и при удвоенной точности не получить верный результат по формуле, приведенной в условии.
2	Реализация соответствующих функций на языке СИ в режиме удвоенной точности (для ввода и вывода использовать функции **fscanf** и **fprintf**). Сравнение С-результаты с ФОРТРАН-результатами.

Работы от **0a** по **2** выполняются в первом семестре на **ФОРТРАНе-77** и **C** (компиляторы g77 и gcc). Работы **3a–3f**, **4a–4d**, **5a–5f** (см. раздел "Решения лабораторных работ (часть II)) выполняются во втором семестре (февраль, март) и касаются возможностей **ФОРТРАНа-95**, частично **C++** и **GNU**-совмещенного программи-

рования (компиляторы **gfortran** и **g++**).
Работы невозможно выполнить успешно без понимания тем, отрабатываемых на задачах домашних заданий. Поэтому решения последних важно сдавать по мере их условий, регулярно, не накапливая горы непонятого материала.

Для решения лабораторных работ необходимо уметь

1. Создавать элементарный **make**-файл.
2. Четко формулировать недостатки полученного исходного текста.
3. Находить пределы табулируемых функций.
4. Грамотно обнаруживать опасные свойства формул, в предлагаемых задачах.
5. Преобразовывать исходные формулы к виду удобному для расчета на ЭВМ.
6. Правильно оформлять запись функций и подпрограмм в отдельных файлах.
7. Грамотно описывать интерфейс функций и подпрограмм.
8. Оформлять ввод числовых данных из файла и их вывод в файл
9. Формировать элементарные **gnuplot**-скприпты.
10. Включать в **make**-файл псевдоцели обзора результатов в виде таблицы и графиков, генерируемых утилитой **gnuplot**.

В задачах лабораторных работ запись ФОРТРАН-функций, предлагаемых в условии, иногда содержит рабочие переменные, которые хранят промежуточные результаты, хотя можно было бы обойтись без их отдельного запоминания. В последнем случае конечный результат даже более близок к правильному, так как промежуточные значения хранятся на многоразрядных регистрах процессора. Тем не менее в реальных расчетах, когда алгоритм записывается целым набором формул, без вспомогательных переменных обычно не обойтись. Так что их наличие в исходных ФОРТРАН-текстах просто моделирует реальную ситуацию, позволяя проявить эффекты, которые часто сглаживаются и маскируются многоразрядностью.

В приложении **I** приводится 30 вариантов задач. Ниже даются примеры возможных решений одного из вариантов (задача N 30). Исходные формулы для большинства задач взяты из учебников, задачников и пособий по математическому анализу.

Приводимый список литературы содержит названия книг, которые использовались при составлении пособия и по которым можно более подробно ознакомиться с тем или иным вопросом, затронутым в пособии.

Автор искренне признателен В.Б.Титову за полезные замечания, советы и обсуждения.

Работа поддержана грантом Президента РФ по поддержке ведущих научных школ НШ-1318.2008.2.

2 Решения лабораторных работ (часть I).

2.1 Условие задачи.

В некоторой задаче потребовалась табулировать функцию

$$\mathbf{w(x)} = \frac{\cos \mathbf{x} - \mathbf{1} + \dfrac{\mathbf{x^2}}{\mathbf{2}} - \dfrac{\mathbf{x^4}}{\mathbf{24}} + \dfrac{\mathbf{x^6}}{\mathbf{720}}}{\dfrac{\sin \mathbf{x}}{\mathbf{x}} - \mathbf{1} + \dfrac{\mathbf{x^2}}{\mathbf{6}} - \dfrac{\mathbf{x^4}}{\mathbf{120}}}$$

для $x = e^{-t}$ при $t \in [1.0, 2.0]$. Была составлена подпрограмма-функция $\mathbf{w0(x)}$:

```
      function w0(x)                                    !    Файл w0.for
      x2=x*x
      a=cos(x) - 1 + x2* 0.5*(1 - x2/12 * ( 1 - x2 / 30))
      b=sin(x)/x - 1+x2/6 * (1-x2/20*(1-x2/42))
      w0=a/b
      end
```

которая вела расчет непосредственно по формуле из заголовка. Тестирование функции проводилось программой

```
      program tsfs2p30                        !    Файл  tsfs2p30.for
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn
      read(ninp,101) n
      write(nres, *) ' #   t0=',t0,'    tn=',tn,'    n=',n
      pi4=atan(1.0)
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
        t=(t0+i*ht)
        x=exp(-t)
        r0=w0(x)
        write(nres,1001) i, t, r0
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'t',14x,'w0')
 1001 format(1x,i5,2x,2x,e15.7,e15.7)
      end
```

и при значениях аргумента $\mathbf{t}$ из диапазона $\mathbf{t} = [-\mathbf{1.0}, -\mathbf{2.0}]$ (т.е. при $\mathbf{x} = \mathbf{e}$ и $\mathbf{x} = \mathbf{e^2}$) дало на одинарной точности результаты верные в пределах семи значащих цифр мантиссы. Табулирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{1.0}, \mathbf{2.0}]$ привело к следующему:

```
#   t0=   1.000000         tn=   2.000000         n=          10
#  i            t              w0
   0      0.1000000E+01  0.5782932E+01
   1      0.1100000E+01  0.6694451E+01
   2      0.1200000E+01  0.6426376E+01
   3      0.1300000E+01  0.7200233E+01
   4      0.1400000E+01  0.3847253E+01
   5      0.1500000E+01  0.1791706E+01
   6      0.1600000E+01  0.3489712E+01
   7      0.1700000E+01  0.3166201E+01
   8      0.1800000E+01  0.3158754E+01
   9      0.1900000E+01  0.2960747E+01
  10      0.2000000E+01  0.3008335E+01
```

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Аналитически найти предел $\lim\limits_{x \to 0} w(x) = ?$
4. Преобразовать расчетную формулу так, чтобы результат был верен.
5. Написать соответствующую новой схеме расчета функцию **w1(x)**.
6. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
7. Включить в **make**-файл вывод рисунка с графиками **w0(x)** и **w1(x)**.
8. Модифицировать программу для получения результатов с удвоенной точностью и проанализировать их.

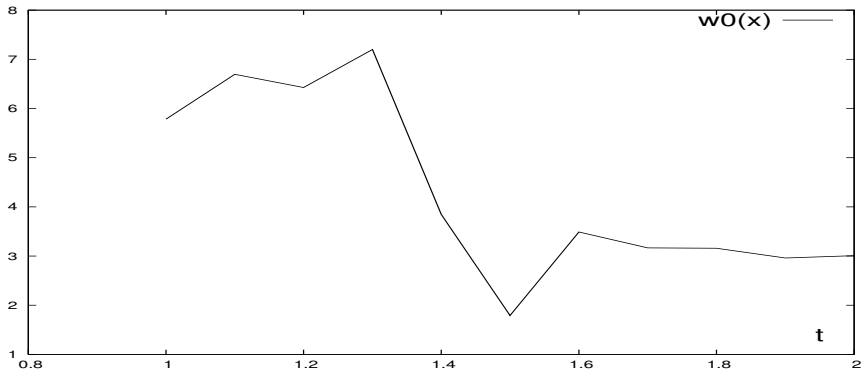

Рис. 1: Графики функции, табулированной алгоритмом w0(x) при $x = e^{-t}$.

2.2 Исходное ФОРТРАН-решение (ЛР: 0a-0c).

1. Первый недостаток полученного исходного текста.
2. Второй и главный недостаток предложенного решения.
3. Объективная и субъективная причины неверности результата.
4. Новая подпрограмма-функция расчета искомого отношения.
5. Make-файл модернизированной программы.
6. Результаты пропуска модернизированной программы.
7. О чем узнали из параграфа 2.2 ?

2.2.1 Первый недостаток полученного исходного текста

Это – использование правила умолчания ФОРТРАНа.

1. Его можно оправдать (с некоторой натяжкой) в небольших программах, когда опечатки в именах легко обнаружить.

2. Так при наборе программы вместо оператора $\mathbf{t = t0 + (i - 1) * h}$ могли бы записать $\mathbf{t = tO + (i - 1) * h}$, не заметив, что вместо цифры нуль в имени переменной **t0** случайно нажали литеру **O** большое. Компиляция программы завершится удачно и при пропуске даже получится некий числовой результат:

```
#   t0=  1.    tn=  2.    n= 10
#  i            t              w0
   0      0.2970153E-38  0.8981886E+01
   1      0.1000000E+00  0.8986946E+01
   2      0.2000000E+00  0.8958036E+01
   3      0.3000000E+00  0.8969361E+01
   4      0.4000000E+00  0.8898288E+01
   5      0.5000000E+00  0.8972149E+01
   6      0.6000000E+00  0.8877596E+01
   7      0.7000000E+00  0.9807290E+01
   8      0.8000000E+00  0.8068257E+01
   9      0.9000000E+00  0.7077634E+01
  10      0.1000000E+01  0.5782932E+01
```

3. Он отличен от предыдущего. Если не заметить, что значение аргумента меньше нужного на единицу, и убедиться, что $\mathbf{w0(1) = 8.981886}$, то чисто психологически можно долго верить в правильность работы программы и функции **w0(x)**.

4. Отмена действия правила умолчания посредством оператора **implicit none** выявит опечатку еще на этапе компиляции и не позволит бессмысленно тратить машинное время. Поэтому всегда явно отменяем действие правила умолчания.

5. Обычно компиляторы при вызове позволяют указать соответствующую опцию. Так вызов GNU-компилятора **g77** с опцией **-Wimplicit** автоматически отключает действие правила умолчания. Однако, соответствующая опция GNU-компилятора **gfortran** с языка ФОРТРАН-95 обозначается иначе: **-fimplicit-none**. Так что проще в каждой программной единице явно указать оператор **implicit none**, чем помнить для разных компиляторов соответствующие длинноязычные опции.

2.2.2 Второй и главный недостаток предложенного решения.

(мое отношение к полученному результату)

В исходной формуле в числителе находится разность между функцией $\cos \mathbf{x}$ и ее приближения первыми четырьмя членами разложения в ряд Маклорена. При $\mathbf{x} \approx \mathbf{0.01}$ значения функции и ее приближения совпадут с точностью до восьми значащих цифр мантиссы так, что операция вычитания на четырехбайтовых ячейках (одинарная точность) приведет к потере чуть ли не всех верных цифр результата.

Подобная ситуация наблюдается и в знаменателе: из функции $\frac{\mathbf{sinx}}{\mathbf{x}}$ вычитаются три старших члена ее разложения с аналогичной потерей верных цифр разности. Поэтому результаты, полученные функцией **w0(x)** – абсолютно неверны, хотя и кажутся правдоподобными. Убедиться в этом можно и аналитически, и практически.

Аналитический подход требует нахождения предела вычисляемой функции при стремлении аргумента к нулю; практический – перехода на удвоенную точность. Последний менее хлопотен и, безусловно, работает в нужном направлении. Однако, имеем ввиду, что и удвоенная точность может оказаться недостаточной для обеспечения правильности старших цифр результата.

ФОРТРАН-решение задачи, приведенное в условии, базировалось на использовании правила умолчания, которое в предыдущем пункте **2.2.1** отмечено как серьезный недостаток. Для его исправлении придется не только отменить действие правила умолчания, но и явно описать типы используемых функций и переменных, а также снабдить запись вещественных числовых констант признаком удвоенной точности.

Сейчас наша цель – **при минимуме изменений работающей исходной программы** поскорее увидеть результат ее работы в режиме **real*8**. Поэтому, отложив более объемные по размеру исправления на потом, просто настроим правило умолчания на режим удвоенной точности, добавив после заголовков главной программы и функции по строке с оператором **implicit real** $*\,\mathbf{8}(\mathbf{a}-\mathbf{h},\mathbf{o}-\mathbf{z})$ для того, чтобы компилятор всем вещественным переменным в обоих файлах сопоставил тип **real*8**.

```
#   t0=   1.00000000000000          tn=   2.00000000000000          n=          10
#  i            t              w0
    0      0.1000000E+01  0.8997541E+01
    1      0.1100000E+01  0.8997986E+01
    2      0.1200000E+01  0.8998351E+01
    3      0.1300000E+01  0.8998650E+01
    4      0.1400000E+01  0.8998895E+01
    5      0.1500000E+01  0.8999095E+01
    6      0.1600000E+01  0.8999259E+01
    7      0.1700000E+01  0.8999393E+01
    8      0.1800000E+01  0.8999503E+01
    9      0.1900000E+01  0.8999594E+01
   10      0.2000000E+01  0.8999671E+01
```

Как видим, расчет, использующий переменные удвоенной точности, дал результаты принципиально отличные от результатов одинарной. Появилась надежда, что новый результат более верен, но, если переход на удвоенную точность по каким-то причинам для заказчика неприемлем, то необходимо искать новую формулу для расчета.

2.2.3 Объективная и субъективная причины неверности результата

Объективная причина получения неверного результата на одинарной точности заключается в недостаточной для используемой расчетной формулы разрядности ЭВМ. Субъективная причина состоит в том, что, зная про наличие объективной причины, мы, не приводя исходную формулу к виду, более подходящему для расчета на одинарной точности, пытаемся посредством операции вычитания найти разность, все значащие цифры которой получаются из разрядов уменьшаемого и вычитаемого, находящихся вне действующей разрядной сетки ЭВМ.

Объективную причину можно устранить, используя специальное математическое обеспечение (например, приложения типа *Математика*, позволяющие вести расчет с гигантским количеством значащих цифр). Упомянутые средства вряд ли подходят для ведения многозначных массовых расчетов при решении задач вычислительного характера из-за неприемлемых временных затрат, хотя полезны и удобны для выборочного высокоточного расчета. Без аппаратной поддержки ячеек сверхвысокой разрядности объективная причина не устраняется.

Субъективную причину можно устранить, если найти такое математическое преобразование, которое позволит аналитически исключить из исходной формулы вычитание почти равных слагаемых, оставив в качестве результата старшие члены разложения самой разности.

В нашем случае такое исключение проводится легко: достаточно просто зачеркнуть в разложениях $\cos\mathbf{x}$ и $\dfrac{\sin\mathbf{x}}{\mathbf{x}}$ вычитаемые в исходной формуле слагаемые. Действительно,

$$\cos\mathbf{x} - \mathbf{1} + \frac{\mathbf{x^2}}{\mathbf{2}} - \frac{\mathbf{x^4}}{\mathbf{24}} + \frac{\mathbf{x^6}}{\mathbf{720}} = \sum_{\mathbf{k=4}}^{\infty}(\mathbf{-1})^{\mathbf{k}}\frac{\mathbf{x^{2k}}}{\mathbf{(2k)!}} = \mathbf{x^8}\sum_{\mathbf{m=0}}^{\infty}(\mathbf{-1^m})\frac{\mathbf{x^{2m}}}{\mathbf{(2m+8)!}}$$

$$\frac{\sin\mathbf{x}}{\mathbf{x}} - \mathbf{1} + \frac{\mathbf{x^2}}{\mathbf{6}} - \frac{\mathbf{x^4}}{\mathbf{120}} = \sum_{\mathbf{k=4}}^{\infty}(\mathbf{-1})^{\mathbf{k}}\frac{\mathbf{x^{2k}}}{\mathbf{(2k+1)!}} = \mathbf{x^8}\sum_{\mathbf{k=0}}^{\infty}(\mathbf{-1^k})\frac{\mathbf{x^{2k}}}{\mathbf{(2k+9)!}}$$

Так что искомая функция выражается отношением

$$\mathbf{w(x)} = \frac{\displaystyle\sum_{\mathbf{k=0}}^{\infty}(\mathbf{-1^k})\frac{\mathbf{x^{2k}}}{\mathbf{(2k+8)!}}}{\displaystyle\sum_{\mathbf{k=0}}^{\infty}(\mathbf{-1^k})\frac{\mathbf{x^{2k}}}{\mathbf{(2k+9)!}}}$$

из которого сразу видно, что точное значение $\mathbf{w(0)=9}$, а при $\mathbf{x=e^{-10}\approx 4.54\cdot 10^{-5}}$ правильное значение $w(e^{-10})$ с точностью чуть ли не до десяти десятичных цифр мантиссы должно совпадать с **девяткой**. При пропуске же варианта с опечаткой **tO** вместо **t0** в режиме удвоенной точности получаем $w(e^{-10}) \approx 8.9975$. Другими словами, в данном случае переход на удвоенную точность хоть и позволил получить результат верный в пределах трех старших цифр мантиссы, тем не менее должен нас насторожить: либо есть ошибка в формуле, либо неудачно выбранная схема расчета.

Вывод: в расчетных формулах не всегда выгодно использование встроенных функций ($\cos x$ и $\sin x$, да и многих других). Полезно знать, понимать и уметь применять альтернативные способы расчета.

2.2.4 Новая подпрограмма-функция расчета искомого отношения.

Новая схема расчета предполагает нахождение двух сумм (для числителя и знаменателя соответственно).

```
function w1(x)   !                              Файл w1.for
x2=x*x           !  Запомнили x**2, чтобы не перевычислять его в цикле.
sa1=1. 0         ! Задаем значения переменных, входящих в условие продолжения
sa=0             ! цикла do while, так, чтобы его тело гарантированно начало
sb=0             ! работу: sa1 - предыдущее значение числителя; sa - текущее.
a=1/40320.0      ! Инициализируем текущее слагаемое  числителя.
b=a/9            ! --"--"--"--"--"--"--"--"--"-- знаменателя.
k=0              ! --"--"--"--"--  индекс (номер) текущего слагаемого.
do while (abs(sa).ne.sa1)   ! Пока не совпали предыдущая и текущая суммы:
  sa1=abs(sa)               !   переопределяем предыдущую на текущую;
  sa=sa+a                   !   уточняем текущую сумму числителя,
  a=-a*x2/(2*k+9)/(2*k+10)  !   вычисляем его очередное слагаемое;
  sb=sb+b                   !   уточняем текущую сумму знаменателя,
  b=-b*x2/(2*k+10)/(2*k+11) !   вычисляем его очередное слагаемое;
  k=k+1                     !   вычисляем номер очередного слагаемого
enddo
w1=sa/sb                    !   присваиваем результат ИМЕНИ ФУНКЦИИ.
end
```

Здесь накопление каждой из сумм реализуется в пределах одного оператора цикла. Уточнять сумму прекращаем как только очередное слагаемое оказывается вне разрядной сетки уже накопленного значения суммы.

Для расчета значения очередного слагаемого выгодна рекурентная формула, которая выражает его через номер и значение предыдущего так, что исключается необходимость явного расчета факториалов и вызова функции возведения в степень:

$$\mathbf{a} = -\mathbf{a} \cdot \frac{\mathbf{x^2}}{(\mathbf{2k}+\mathbf{9})(\mathbf{2k}+\mathbf{10})}$$

$$\mathbf{b} = -\mathbf{b} \cdot \frac{\mathbf{x^2}}{(\mathbf{2k}+\mathbf{10})(\mathbf{2k}+\mathbf{11})}$$

Из формул пункта **2.2.3** видно, что начальное значение слагаемого для числителя равно равным $a = \dfrac{1}{40320.0}$, а для знаменателя $b = \dfrac{1}{362880.0}$.

Замечание 1: Описание функции **w1**, приведенное выше, использует, как и описание функции **w0**, правило умолчания традиционного ФОРТРАНа. Этот недостаток будет исправлен в окончательном варианте.

Замечание 2: Для текущих слагаемых числителя и знаменателя в приведенном тексте **w1** выбраны имена **a** и **b** соответственно. Возможно, целесообразнее было бы предложить **ak** и **bk**. Однако, начинающие, видя в формуле переменную с индексом, зачастую, не задумываясь, автоматически оформляют ее массивом (от $\mathbf{a_k}$ до $\mathbf{a(k)}$ или $\mathbf{a[k]}$ новичку один шаг). Выбранные же **a** и **b**, по крайней мере, могут спровоцировать вопрос:"**Почему они не массивы?**".

2.2.5 Make-файл модернизированной программы.

```
compiler:=gfortran
       main : tsfs2p30a.o w0.o w1.o
[  TAB ]     $(compiler) tsfs2p30a.o w0.o w1.o -o main
tsfs2p30a.o : tsfs2p30a.for
[  TAB ]     $(compiler) -c tsfs2p30a.for
       w0.o : w0.for
[  TAB ]     $(compiler) -c w0.for
       w1.o : w1.for
[  TAB ]     $(compiler) -c w1.for
     clear  :
[  TAB ]     rm *.o
     result : input main
[  TAB ]     ./main
     restab : result main
[  TAB ]     cat result
     resplt : result main
[  TAB ]     gnuplot 'view.gnu'
```

Здесь **compiler** – переменная, придуманная нами, для упрощения модификации **make**-файла на случай замены компилятора. Именно, в переменную **compiler** (в первой строке **make**-файла) посредством оператора присваивания **:=** заносится имя команды вызова компилятора. Так что для смены компилятора достаточно подправить лишь первую строку **make**-файла (например, **compiler:=g77**). Запись **$(compiler)** означает, что утилита **make** должна взять имя команды из переменной **compiler**.

Видим, что добавление новой зависимости в рабочий проект (например, потребуется еще и файл **w2.for**) приведет к необходимости добавления новой цели **w2.o** и команды ее достижения. Такой подход полезен при начальном освоении технологии работы с утилитой **make**, обеспечивая в принципе лучшее понимание необходимости описания новых целей и зависимостей.

На самом деле есть возможность потребовать от утилиты **make** автоматическую формировку имен целей (без явного указания их в тексте **make**-файла). Так что размер **make**-файла может и не зависеть линейно от числа описываемых целей, оставаясь даже для проектов, использующих много исходных ФОРТРАН- или С-файлов, зачастую не больше приведенного выше (подробнее см., например, курс “Операционные системы”).

2.2.6 Результаты пропуска модернизированной программы.

```
#   t0=  -1.000000         tn=  -2.000000         n=           5
#  i            t              w0             w1
   0     -0.1000000E+01  0.8869353E+01  0.8869358E+01
   1     -0.1200000E+01  0.8807819E+01  0.8807821E+01
   2     -0.1400000E+01  0.8719338E+01  0.8719340E+01
   3     -0.1600000E+01  0.8594666E+01  0.8594665E+01
   4     -0.1800000E+01  0.8424662E+01  0.8424664E+01
   5     -0.2000000E+01  0.8205046E+01  0.8205045E+01
```

Убеждаемся, что, если при $\mathbf{t} \in [-\mathbf{2}, -\mathbf{1}]$ результаты, полученные обеими функциями $\mathbf{w0}(\mathbf{x})$ и $\mathbf{w1}(\mathbf{x})$ практически совпадают,

то на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{1}, \mathbf{2}]$

```
#   t0=   1.000000         tn=   2.000000         n=          10
#  i            t              w0             w1
   0      0.1000000E+01  0.5782932E+01  0.8997540E+01
   1      0.1100000E+01  0.6694451E+01  0.8997986E+01
   2      0.1200000E+01  0.6426376E+01  0.8998350E+01
   3      0.1300000E+01  0.7200233E+01  0.8998650E+01
   4      0.1400000E+01  0.3847253E+01  0.8998894E+01
   5      0.1500000E+01  0.1791706E+01  0.8999095E+01
   6      0.1600000E+01  0.3489712E+01  0.8999260E+01
   7      0.1700000E+01  0.3166201E+01  0.8999393E+01
   8      0.1800000E+01  0.3158754E+01  0.8999504E+01
   9      0.1900000E+01  0.2960747E+01  0.8999595E+01
  10      0.2000000E+01  0.3008335E+01  0.8999667E+01
```

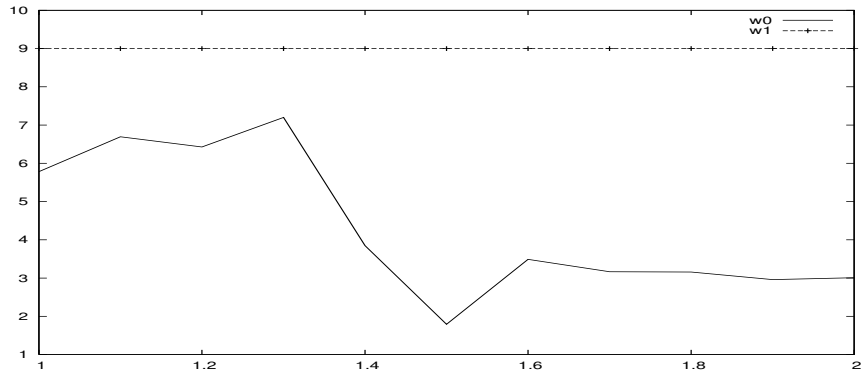

Рис. 2: Графики функции, табулированной w0(x) и w1(x) при $x = e^{-t}$.

и, тем более, при $\mathbf{t} \in [\mathbf{5}, \mathbf{6}]$

```
#   t0=   5.000000         tn=   6.000000         n=          10
#  i            t              w0             w1
   0      0.5000000E+01  0.2999991E+01  0.8999999E+01
   1      0.5100000E+01  0.2999992E+01  0.8999999E+01
   2      0.5200000E+01  0.2999995E+01  0.8999999E+01
   3      0.5300000E+01  0.2999994E+01  0.9000000E+01
   4      0.5400000E+01  0.2999988E+01  0.9000000E+01
   5      0.5500000E+01  0.2999997E+01  0.9000000E+01
   6      0.5600000E+01  0.2999976E+01  0.9000000E+01
   7      0.5700000E+01  0.2999997E+01  0.9000000E+01
   8      0.5800000E+01  0.2999943E+01  0.9000000E+01
   9      0.5900000E+01  0.2999998E+01  0.9000000E+01
  10      0.6000000E+01  0.3000054E+01  0.9000000E+01
```

соответствующие результаты существенно различны: функция $\mathbf{w1}(\mathbf{x})$ при $x \ll 1$ получает правильное значение в отличие от функции $\mathbf{w0}(\mathbf{x})$.

Замечание. Исследование работы функций **w0** и **w1** при отрицательных значениях **t** таких, что $\mathbf{x} \gg \mathbf{1}$ не требуется по условию задачи. Однако, анализ правильности соответствующих результатов не менее полезен для уяснения другой причины потери точности (см., например, третью часть настоящего пособия).

2.2.7 О чем узнали из параграфа 2.2 ?

1. При программировании расчетных формул необходимо помнить, что формулы верные аналитически могут обладать свойствами, которые делают бессмысленным их непосредственное использование для расчета на ЭВМ.

2. Объективная причина возникновения подобных свойств – конечная разрядность ячеек, из-за которой данное вещественного типа представляется машиной лишь приближенно, как правило, с погрешностью отличной от нуля.

3. Величина этой погрешности для вводимого данного обычно гораздо меньше по модулю величины самого данного. Так для данного типа **real*4** она меньше введенного значения примерно в $\mathbf{10^7}$ раз, а для данного типа **real*8** – в $\mathbf{10^{16}}$ раз. Кажется, что подобными погрешностями можно пренебречь.

4. Однако машинный процесс расчета по формуле в силу конечной разрядности ячейки приводит к распространению начальных погрешностей, а иногда и к их катастрофическому росту так, что погрешность результата расчета может на много порядков превысить значение самого результата.

5. Для сильной потери точности вовсе не требуется гигантское число арифметических операций. Так все верные цифры результата теряются за одну операцию вычитания почти равных чисел, то есть чисел различающихся лишь несколькими младшими битами разрядной сетки. Например, достаточно посмотреть на результат работы фрагмента ФОРТРАН-программы:

```
write(*,*) 1.2345678-1.2345677  !  который равен  1.1920929E-07
```

6. Поэтому, прежде чем программировать ту или иную формулу для расчета, необходимо проанализировать ее на выявление условий и операций, которые могут привести к резкой потере точности.

7. После их выявления следует постараться аналитически преобразовать формулу так, чтобы исключить из нее опасные для вычисления на ЭВМ фрагменты.

8. При вычитании почти равных величин, когда вычитаемое представляет собой извлечение кваратного корня, часто помогает умножение и деление формулы на соответствующее сопряженное выражение с последующим аналитическим исключением равных величин, не поручая вычитание ЭВМ.

9. При наличии в формуле вычитания почти равных тригонометрических выражений, можно, используя подходящие формулы, опять-таки, постараться исключить вычитание аналитически.

10. Стандартные приемы раскрытия неопределенностей при вычислении пределов часто нередко получают формулу аналитически эквивалентную исходной, но свободную от влияния эффектов, связанных с потерей точности. Другими словами, эти приемы имеют не только академическое, но и ценное прикладное значение.

2.3 Решения задачи в стиле ФОРТРАНа-77 (ЛР 1a, 1b).

1. Одинарная точность (лабораторная 1a).
2. Удвоенная точность (лабораторная 1b).
*3. Результаты тестирования функций w0 и w1 типа real*8.*
4. О чем узнали из параграфа 5.3? (кратко)

2.3.1 Одинарная точность (лабораторная 1a).

```
      program tsfs2p3077                      !    Файл  tsfs2p3077.for
      implicit none
      real w0, w1, t0, tn, t, ht, x, r0, r1
      integer ninp, nres, n, i
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn; read(ninp,101) n
      write(nres, *) ' #   t0=',t0,'    tn=',tn,'    n=',n
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n;      t=(t0+i*ht); x=exp(-t); r0=w0(x); r1=w1(x)
                     write(nres,1001) i, t, r0, r1
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'t',14x,'w0',14x,'w1')
 1001 format(1x,i5,2x,2x,e15.7,e15.7,e15.7)
      end

      function w0(x)                                !    Файл w0.for
      implicit none
      real x, x2, w0, a, b
      x2=x*x; a=cos(x) - 1 + x2* 0.5*(1 - x2/12 * ( 1 - x2 / 30))
              b=sin(x)/x - 1+x2/6 * (1-x2/20*(1-x2/42))
      w0=a/b
      end
      function w1(x)                                !    Файл w1.for
      implicit none
      real w1, x, x2, sa1, sa, sb, a, b
      integer k
      x2=x*x; sa1=1.0; sa=0; sb=0; a=1/40320.0; b=a/9
      k=0
      do while (abs(sa).ne.sa1)
        sa1=abs(sa); sa=sa+a; a=-a*x2/(2*k+ 9)/(2*k+10)
                     sb=sb+b; b=-b*x2/(2*k+10)/(2*k+11); k=k+1
      enddo
      w1=sa/sb
      end
```

Заметим, что в подпрограмме-функции **w1**(**x**) начальные значения сумм опасно задавать оператором **data**, так как при повторных обращениях **w1**(**x**) новый результат будет прибавляться к предыдущему в силу того, что оператор **data** – невыполняемый (работает на этапе компиляции, но не на этапе выполнения программы).

2.3.2 Удвоенная точность (лабораторная 1b).

```
      program tsfd2p3077                        !    Файл  tsfd2p3077.for
      implicit none
      real*8 wd0, wd1, t0, tn, t, ht, x, r0, r1
      integer ninp, nres
      integer n, i
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn
      read(ninp,101) n
      write(nres, 1099) t0, tn, n
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
        t=(t0+i*ht); x=dexp(-t); r0=wd0(x); r1=wd1(x)
        write(nres,1001) i, t,  r0, r1
      enddo
      close(nres)
  100 format(d15.7)
  101 format(i15)
 1099 format(1x,' #   t0=',d23.15,2x,'tn=',d23.15,2x,'n='i3)
 1100 format(1x,' #',2x,'i',14x,'t',21x,'w0',21x,'w1')
 1001 format(1x,i5,2x,d23.15,d23.15,d23.15)
      end

      function wd0(x)                                  !    Файл wd0.for
      implicit none
      real*8 wd0, x, x2, a, b
      x2=x*x
      a=dcos(x) - 1 + x2* 0.5d0*(1 - x2/12d0 * ( 1 - x2 / 30d0))
      b=dsin(x)/x - 1d0+x2/6d0 * (1-x2/20d0*(1-x2/42d0))
      wd0=a/b
      end


      function wd1(x)                                  !    Файл w1.for
      implicit none
      real*8 wd1, x, x2, sa1, sa, sb, a, b
      integer k, k2
      x2=x*x
      sa1=1d0; sa=0d0; sb=0d0; a=1d0/40320d0; b=a/9d0
      k=0
      do while (abs(sa).ne.sa1)
        sa1=abs(sa); k2=2*k
        sa=sa+a; a=-a*x2/(k2+ 9)/(k2+10)
        sb=sb+b; b=-b*x2/(k2+10)/(k2+11)
        k=k+1
      enddo
      wd1=sa/sb
      end
```

1. Заметим, что имена используемых встроенных функций начинаются с буквы **d**, а числовые константы заканчиваются суффиксом **d0**. В первом случае литера **d** условно напоминает, что встроенная функция, получает результат с удвоенной точностью, а не с одинарной; во втором – она однозначно указывает на задание значения константы вещественного типа с удвоенной точностью. При использовании прежней формы вызова встроенных функций компиляция на старых компиляторах могла завершиться выдачей сообщения о том, что, например, встроенная функция **cos**(**x**) должна иметь аргумент типа **real** $*$ **4**, но не **real** $*$ **8** (с последним должна работать только функция с именем **dcos**).

2. В современном ФОРТРАНе имена встроенных функций **cos**(**x**) и **dcos**(**x**) часто называют **специфическими** [2], подчеркивая специфическую настройку первой на работу с с одинарной точностью, а второй – с удвоенной.

3. Наряду со **специфическими** именами функций в ФОРТРАНе-95 существует понятие **родового** имени функции, которое обычно совпадает с ее общепринятым математическим обозначением (например, **cos**), но позволяет автоматически в зависимости от типа аргумента вызывать разные алгоритмы расчета значений функции $\cos \mathbf{x}$: при аргументе типа **real** $*$ **4**, вызывается функция со **специфическим** именем **cos**, а при аргументе типа **real*8** вызывается функция со **специфическим** именем **dcos**. Современный ФОРТРАН предоставляет также возможность сопоставлять одному **родовому** имени в зависимости от типа и/или количества аргументов различные алгоритмы, каждый из которых, именуется своим **специфическим** именем.

4. Наличие суффикса **d** при записи константы вещественного типа в тексте программы на ФОРТРАНе-77 является обязательным. Запись порядка с суффиксом **d** – это тот *клочок сена*, который легко подложить и который гарантирует, что константа будет задана с удвоенной точностью. Попробуйте, например, пропустить программу:

```
write(*,'(d25.16)') 1.25
write(*,'(d25.16)') 1.3
write(*,'(d25.16)') 1.3d0   !  и в ФОРТРАНе-77, и в ФОРТРАНе-95
write(*,'(d25.16)') 1.3_8   !   только начиная с ФОРТРАНа-95 !!!
end
```

5. Обратим внимание на изменение в программе формата выводимых вещественных данных (раньше было **e15.7**, а теперь **d23.16**. Если количество позиций отводимых на вывод числа обозначить **w**, а количество цифр после запятой, через **d**, то при формировке дескриптора формата с плавающей запятой следует придерживаться правила **w**$>$**d**$+$**7**, так как обычно на порядок числа отводится две позиции, по одной позиции на: буквы **e** (или **d**), десятичную точку, ведущий нуль в целой части числа, возможный знак числа и на пробел, разделяющий числа (2+1+1+1+1+1=7).

2.3.3 Результаты тестирования функций w0 и w1 типа real*8.

```
#   t0= -0.200000000000000E+01  tn= -0.100000000000000E+01  n= 10
#  i             t                    w0                    w1
   0   -0.200000000000000E+01  0.820504574900117E+01  0.820504574900117E+01
   1   -0.190000000000000E+01  0.832090869104376E+01  0.832090869104376E+01
   2   -0.180000000000000E+01  0.842466218276487E+01  0.842466218276487E+01
   3   -0.170000000000000E+01  0.851579733889685E+01  0.851579733889685E+01
   4   -0.160000000000000E+01  0.859466581258819E+01  0.859466581258819E+01
   5   -0.150000000000000E+01  0.866213615572492E+01  0.866213615572491E+01
   6   -0.140000000000000E+01  0.871933992340798E+01  0.871933992340798E+01
   7   -0.130000000000000E+01  0.876749986560483E+01  0.876749986560483E+01
   8   -0.120000000000000E+01  0.880782229096249E+01  0.880782229096248E+01
   9   -0.110000000000000E+01  0.884143534014924E+01  0.884143534014922E+01
  10   -0.100000000000000E+01  0.886935813470491E+01  0.886935813470491E+01

#   t0=  0.100000000000000E+01  tn=  0.200000000000000E+01  n= 10
#  i             t                    w0                    w1
   0    0.100000000000000E+01  0.899754059915270E+01  0.899754060051501E+01
   1    0.110000000000000E+01  0.899798622354877E+01  0.899798622967396E+01
   2    0.120000000000000E+01  0.899835115227924E+01  0.899835114074282E+01
   3    0.130000000000000E+01  0.899864992248471E+01  0.899864994539330E+01
   4    0.140000000000000E+01  0.899889457760761E+01  0.899889461325594E+01
   5    0.150000000000000E+01  0.899909497495678E+01  0.899909494866309E+01
   6    0.160000000000000E+01  0.899925903870764E+01  0.899925898169114E+01
   7    0.170000000000000E+01  0.899939264655231E+01  0.899939328880004E+01
   8    0.180000000000000E+01  0.899950319812463E+01  0.899950325567335E+01
   9    0.190000000000000E+01  0.899959372813654E+01  0.899959329262975E+01
  10    0.200000000000000E+01  0.899967054454032E+01  0.899966701113196E+01

#   t0=  0.200000000000000E+01  tn=  0.200000000000000E+02  n=  9
#  i             t                    w0                    w1
   0    0.200000000000000E+01  0.899967054454032E+01  0.899966701113196E+01
   1    0.400000000000000E+01  0.786754041065968E+01  0.899999390068712E+01
   2    0.600000000000000E+01 -0.203909920048154E+01  0.899999988828705E+01
   3    0.800000000000000E+01  0.111588938446488E+01  0.899999999795391E+01
   4    0.100000000000000E+02 -0.134166958739005E+01  0.899999999996252E+01
   5    0.120000000000000E+02  0.129779789700786E+00  0.899999999999931E+01
   6    0.140000000000000E+02 -0.255347294269858E+01  0.899999999999999E+01
   7    0.160000000000000E+02 -0.551127146827302E+00  0.900000000000000E+01
   8    0.180000000000000E+02  0.339011001966861E+00  0.900000000000000E+01
   9    0.200000000000000E+02  0.300000000000000E+01  0.900000000000000E+01
```

2.3.4 О чем узнали из параграфа 2.3? (кратко)

1. Всегда полезно в каждой программной ФОРТРАН-единице явно отключать правило умолчания посредством оператора **implicit none**.

2. Хотя оператор ФОРТРАНа **data** и называется оператором задания начальных значений, важно помнить, что он выполняется на этапе компиляции, но не выполнения программы. Так что рабочие ФОРТРАН-переменные функций и подпрограмм *во время их повторных вызовов* не инициализируются посредством **data**, а сохраняют значения, записанные в них при предыдущем обращении.

3. В СИ и С++ все локальные переменные, то есть описанные внутри функций, по умолчанию являются *автоматическими*. В частности, это означает, что при очередном вызове функции явная инициализация такой переменной будет происходить заново при каждом новом вызове функции, а не при компиляции, как в случае оператора **data** ФОРТРАНа. Поэтому, при ручном переводе текста С-функции на язык ФОРТРАН, требуемую инициализацию в ФОРТРАНе (если функция изменяет значение инициализированной переменной), проводим НЕ оператором **data**, НЕ возможным продолжением оператора описания переменной, которое нацелено на инициализацию (например, **real*8 a /1.4d0/** или **real(8):: a=1.4d0**), а простым отдельным выполняемым **оператором** присваивания: **a=1.4**.

4. В современном ФОРТРАНе есть понятие **родового имени** функции, по которому можно вызвать любую из некоторого семейства функций, отличающихся типом и/или количеством аргументов. У каждой из таких функций есть и свое **специфическое имя**. Механизм, обеспечивающий возможность по одному родовому имени вызывать разные функции называется **перегрузкой функций**.

5. В старых версиях ФОРТРАНа запись в текстах программ числовых констант с удвоенной точностью с необходимостью требовала наличия порядка с использованием литер **d** или **D** перед порядком числа, например,

```
1.8d0
```

6. Современный ФОРТРАН допускает и форму записи без порядка, например,

```
1.8_8
```

где после подчеркивания указывается параметр разновидности (4 – для одинарной точности; 8 – для удвоенной).

2.4 Решение контрольной на СИ (ЛР 2a, 2b).

1. Чуть-чуть о вводе данных из файла и выводе в файл.
2. Одинарная точность (лабораторная 2a).
3. Удвоенная точность (лабораторная 2b).
4. О чем узнали из параграфа 2.4? (кратко)

2.4.1 Чуть-чуть о вводе данных из файла и выводе в файл.

Ввод исходных данных в С-программе можно осуществить через устройство стандартного ввода **stdin**, перенастроив его на ввод из одного конкретного файла посредством операции перенаправления (<) операционной системы. Язык С не имеет встроенных операторов ввода-вывода, хотя и обладает широким набором соответствующих библиотечных функций. Доступ к ним обеспечивается подключением к программе через оператор **include** файла <**stdio.h**>. Так форматизованный ввод с экрана выполняется функцией **scanf**, а вывод на экран – функцией **printf**.

При решении расчетных задач исходные данные обычно удобно хранить в файле. В СИ для форматизованного ввода числовых данных из файла имеется функция **fscanf**, а для форматизованного вывода в файл – функция **fprintf**. Перед их вызовом необходимо, чтобы соответствующий файл был открыт. Открытие файла осуществляется функцией **fopen**, которой на вход подаются два аргумента:

1. имя файла, по которому операционнная система сможет его распознать;
2. способ доступа к файлу (чтение, запись; их режимы)

При удачном открытии файла функция **fopen** возвращает через свое имя значение указателя на файл. Этот указатель указывает на предопределенную в С структуру типа **FILE**. Описание **FILE *fp;** означает, что переменная **fp** предназначена для хранения указателя на файл. Правда, пока содержимое этой переменной неизвестно.

```
#include <stdio.h>
int main()
{FILE *fpi, *fpr; int a; float b, c; double d;
  fpi=fopen("input","r");
  if (fpi==NULL) { printf("Неудача при открытии файла input !!!\n"); return 1;}
  fpr=fopen("result","w");
  if (fpr==NULL){ printf("Неудача при открытии файла result !!!\n"); return 2;}
   fscanf(fpi,"%i %f %e %le", &a, &b, &c, &d);
   fprintf(fpr,"a=%d  b=%f  c=%e   d=%25.15le\n",a, b, c, d);
   fclose(fpr);
   fclose(fpi);
   return 0;
}
```

Здесь

1. **fpi** - указатель на файл с именем **input** (предполагается, что в этот файл пользователь поместит значения параметров, вводимых программой).
2. **fpr** - указатель на файл с именем **result** (предполагается, что в **result** программа будет выводить результаты).

3. Второй параметр в обращении к функции **fopen** задает способ доступа к файлу. Их несколько. Ограничимся пока тремя:

Код доступа к файлу	Смысловая нагрузка
r	Открытие потока (файла) для чтения.
w	Открытие пустого потока для записи.
a	Открытие потока для записи в конец файла (если файла нет, то он создается)

4. Если файл **input** с исходными данными

```
1 3.4 3.5e1 7.1e2
```

присутствует в текущей директории, то после пропуска данной программы получим в ней же результирующий файл с именем **result**:

```
a=1  b=3.400000  c=3.500000e+01   d=    7.100000000000000e+02
```

5. Если же файла с именем **input** в текущей директории нет (например, забыли его создать или назвали не именем **input**), то в качестве результата на экран высветится фраза

```
Неудача при открытии файла input !!!
```

Аналогичное сообщение поступит при открытии файла **result** на запись, если диск переполнен или предпринята попытка записи в файл несуществующей директории (например, если в текущей директории нет подкаталога **absent**, а в качестве первого аргумента при вызове функции **fopen** использована строка **"./absent/result"**.

6. Признаком того, что открытие файла прошло неудачно служит нулевое значение указателя, возвращаемое через имя **fopen**. Поэтому прежде чем начинать чтение или запись следует убедиться, что открытие потока (файла) действительно состоялось. В данной программе проверка выполнена сравнением значения указателя, выработанного функцией **fopen**, с нулевым указателем (константа **NULL**). Впрочем, условие можно записать и короче, например:

$$if(!fpi)\{printf(\text{“Неудача при открытии } input\backslash n'');\, return 1;\}$$

7. На древних компиляторах и операционных системах бывало, что после завершения программы при отсутствии явного закрытия файла с результатом конечная часть результата физически не передавалась из буфера вывода в файл на диске.

Современные системы в принципе берут *закрытие незакрывшихся файлов* окончившихся заданий на себя. Тем не менее, признаком хорошего стиля программирования считается явный вызов оператора или функции закрытия файла. Поэтому
Не забываем явно закрывать файлы с результатом!

2.4.2 Одинарная точность (лабораторная 2а).

С учетом изложенного в предыдущем пункте приведем СИ-программу, решающую рассматриваемую задачу

```
#include <stdio.h>         //  Файл  tsfs2p30.c
#include "w01.h"
int main()
{  float t0, tn, t, ht, x, r0, r1;
   int n, i;
  FILE *ninp, *nres;
  ninp=fopen("input","r");
  if (ninp==NULL) { printf("Неудача при открытии файла input !!!\n"); return 1;}
  nres=fopen("result","w");
  if (nres==NULL){ printf("Неудача при открытии файла result !!!\n"); return 2;}

   fscanf(ninp,"%e %e %d", &t0, &tn, &n);
   fprintf(nres," #  t0=%e  tn=%e  n=%i\n", t0, tn, n);
   ht=(tn-t0)/n;
   fprintf(nres," # %2s %10s %15s %15s\n","i","t","r0","r1");
   for (i=1; i<=n;i++)
     { t=t0+ht*(i-1);
       x=exp(-t);
       r0=w0(x);
       r1=w1(x);
       fprintf(nres,"%5i %15.7e %15.7e %15.7e\n", i, t, r0, r1);
     }
   fclose(ninp);
   fclose(nres);
   return 0;
}

float w0(float x)                                    //    Файл w0.c
{float x2, a, b;
      x2=x*x;
      a=cos(x) - 1 + x2* 0.5*(1 - x2/12 * ( 1 - x2 / 30));
      b=sin(x)/x - 1+x2/6 * (1-x2/20*(1-x2/42));
      return(a/b);
}

float w1(float x)                                     //    Файл w1.c
{float x2, sa1, sa, sb, a, b;
 int k, k2;
 x2=x*x; sa1=1.0; sa=0; sb=0; a=1/40320.0; b=a/9;
 k=k2=0;
 while (fabs(sa)!=sa1)
   { sa1=fabs(sa);
     sa=sa+a; a=-a*x2/(k2+ 9)/(k2+10);
     sb=sb+b; b=-b*x2/(k2+10)/(k2+11);
     k=k+1; k2=2*k;
   }
 return (sa/sb);
}
```

Замечания:

1. **make**-файл к СИ-программе:

```
compiler:=gcc
 main       : tsfs2p30.o w0.o w1.o
[  TAB  ]    $(compiler) tsfs2p30.o w0.o w1.o -lm -o main
 tsfs2p30.o : tsfs2p30.c
[  TAB  ]    $(compiler) -c tsfs2p30.c
 w0.o       : w0.c
[  TAB  ]    $(compiler) -c w0.c
 w1.o       : w1.c
[  TAB  ]    $(compiler) -c w1.c
 clear      :
[  TAB  ]     rm *.o
 result     : input main
[  TAB  ]     ./main
 restab     : result main
[  TAB  ]     cat result
 resplt     : result main
[  TAB  ]     gnuplot 'view.gnu'
```

2. Результат ее работы и для **r0** и **r1** совпадает в пределах семи значащих цифр мантиссы с результатами соответствующей ФОРТРАН-программы.

3. **Помним:** Исходные ФОРТРАН-данные в файле **input** в форме с порядком можно записать и в виде **1.000000e+00**, и в виде **1.000000+00**, иначе говоря, *без литеры "e" !*, что бывает удобно. В СИ последний вариант (*без литеры "e"*) записи значений в файле с исходными данными **недопустим !!!**

4. Если в приведенной программе при описании прототипа функции **w0** указать тип аргумента отличный от типа аргумента, приведенного в описании функции (например, **double**), то когда описания функций находятся в отдельных файлах, С-компилятор не сообщит, как хотелось бы, о несовпадении типов формального и фактического аргументов (компилятор С++ сообщает).

5. Разрешение С-проблемы из пункта 4 в использовании так называемого заголовочного файла (например, myheader.h):

```
float w0(float);       // Файл myheader.h
float w1(float);
```

   Посредством оператора

```
#include "myheader.h"
```

   содержимое файла **myheader.h** можно подсоедить к каждому из файлов, в которых встречается описание нужных функций или обращения к ним. В этом случае контроль гарантирован.

2.4.3 Удвоенная точность (лабораторная 2b).

```
#include <stdio.h>                              //  Файл  tsfd2p30.c
#include "myheader.h"
int main()
{  double t0, tn, t, ht, x, r0, r1;
   int n, i;
  FILE *ninp, *nres;
  ninp=fopen("input","r");
  if (ninp==NULL) {printf("Неудача при открытии файла input !!!\n");return 1;}
  nres=fopen("result","w");
  if (nres==NULL) {printf("Неудача при открытии файла result  !\n");return 2;}
   fscanf(ninp,"%le %le %d", &t0, &tn, &n);
   fprintf(nres," #  t0=%le  tn=%le  n=%i\n", t0, tn, n);
   ht=(tn-t0)/n;
   fprintf(nres," # %2s %10s %17s %23s\n","i","t","r0","r1");
   for (i=1; i<=n;i++)
     { t=t0+ht*(i-1); x=exp(-t);
       r0=w0(x);
       r1=w1(x);
       fprintf(nres,"%5i %15.7e %23.16le %23.16le\n", i, t, r0, r1);
     }
   fclose(ninp); fclose(nres); return 0;
}

#include "myheader.h"
double w0(double x)                                    //    Файл w0.c
{double x2, a, b;
      x2=x*x;
      a=cos(x) - 1 + x2* 0.5*(1 - x2/12 * ( 1 - x2 / 30));
      b=sin(x)/x - 1+x2/6 * (1-x2/20*(1-x2/42));
      return(a/b);
}

#include "myheader.h"
double w1(double x)                                     //    Файл w1.c
{double x2, sa1, sa, sb, a, b;
 int k, k2;
 x2=x*x; sa1=1.0; sa=0; sb=0; a=1/40320.0; b=a/9;
 k=k2=0;
 while (fabs(sa)!=sa1)
   { sa1=fabs(sa);
     sa=sa+a; a=-a*x2/(k2+ 9)/(k2+10);
     sb=sb+b; b=-b*x2/(k2+10)/(k2+11);
     k=k+1; k2=2*k;
   }
 return (sa/sb);
}



double w0(double);                              // Файл   myheader.h
double w1(double);
```

2.4.4 О чем узнали из параграфа 2.4? (кратко)

1. Несмотря на наличие операции перенаправления потоков стандартных ввода и вывода, полезно уметь организовывать и ввод, и вывод средствами языка программирования.

2. В языке СИ для форматного ввода числовых данных из файла служит функция **fscanf**, а для аналогичного вывода функция **fprintf**. В отличие от функций **scanf** и **printf** их первым аргументом служит переменная, хранящая указатель на соответствующий файл.

3. Тип этой переменной при описании задается служебным словом **FILE**.

4. Для инициализации ее содержимого, т.е. для помещения в нее указателя на реальный файл операционной системы (точнее для открытия потока и связывания его с конкретным файлом на диске), служит функция **fopen**, которая имеет два аргумента типа **указатель** на строку: первый содержит имя файла (или путь к нему), второй – задает режим использования.

5. Если в процессе открытия файла обнаружится ошибка (например, нет места диске или опечатка в строке, задающей имя файла), то функция **fopen** возвратит в качесте результата указатель **NULL**.

6. Поэтому, прежде чем читать из файла или писать в файл полезно убедиться, что открытие завершено корректно (что указатель возвращаемый **fopen** не равен **NULL**).

7. В остальном работа с **fscanf** и **fprintf** аналогична работе с **scanf** и **printf**.

8. Перед завершением работы программы открытый файл **необходимо** закрыть. Закрытие файла открытого **fopen** осуществляется функцией **fclose**, которая выгружает содержимое буфера вывода в файл и разрывает связь с потоком [9]. Если закрытие произошло успешно, то **fclose** возвращает **NULL**.

9. Если программа состоит из большого количества файлов, хранящих описания каких-то функций, в которых, возможно, есть обращения к функциям, описанным в других файлах, то полезно интерфейсную часть этих функций (описание прототипов) поместить в заголовочный файл, подсоединяемый к нужным программным единицам посредством оператора

   ```
   #include "имя header-файла.h"
   ```

   Это обеспечит автоматический контроль типов аргументов функций.

3 Решения лабораторных работ (часть II)

4 Приложение I

4.1 Расчет числа π.

Некто с целью проверки значения константы $\pi = \mathbf{3.14159265358979}$, приводимое в справочниках, составил программу расчета набора последовательных приближений длины полуокружности радиуса , приближая последнюю длиной полупериметра правильного вписанного **n**-угольника, где $\mathbf{n = 6 \cdot 2^k}$ при $\mathbf{k = 0, 1, 2 \cdots, 14}$. Расчет полупериметра велся по формуле

$$\mathbf{p_k = n * b_k}$$

, где для вычисления $\mathbf{b_k}$ (половины длины стороны очередного **n**-угольника) использовалось ее выражение через $\mathbf{b_{k-1}}$ (половину длины стороны предыдущего):

$$b_k = \frac{\sqrt{r \cdot (r - \sqrt{r^2 - b_{k-1}^2})}}{2}.$$

Тогда соответствующее приближенное значение π дается формулой $\pi \approx \frac{p_k}{r}$
Была написана следующая программа:

```
      program tsfs2p01                          !     Файл tsfs2p01.for
      data ninp / 5 /
      data nres / 6 /
      open(unit=ninp,file='input')
      open(unit=nres,file='result')
      pi=4.0*atan(1.0)
      write(nres,*) ' #  pi=',pi
      read(ninp,101) r
      write(nres,*) ' #   r=',r
      write(nres,1000)
      dn=6
      b=r*0.5
      p=3*r
      do i=0,14
        write(nres,1001) i,dn, b, p/r
        b=half(r,b)
        dn=dn*2
        p=dn*b
      enddo
c                                        Форматы ввода-вывода:
  101 format(e10.3)
 1000 format(1x,' #',2x,'i',12x,'n',10x,'b(n)',10x,'p(n)/r')
 1001 format(1x,i5,2x,e15.7,e15.7,e15.7)
      end

      function half(r,x)                    !   Файл  half.for
      x2=x*x
      r2=r*r
      half=sqrt( r*(r-sqrt(r2-x2))*0.5 )
      end
```

которая в результате пропуска при $\mathbf{r = 1.5}$ дала такой результат:

```
#  pi=   3.141593
#   r=   3.000000
#  i            n          b(n)          p(n)/r
   0    0.6000000E+01  0.1500000E+01  0.3000000E+01
   1    0.1200000E+02  0.7764573E+00  0.3105829E+01
   2    0.2400000E+02  0.3915789E+00  0.3132631E+01
   3    0.4800000E+02  0.1962089E+00  0.3139342E+01
   4    0.9600000E+02  0.9815729E-01  0.3141033E+01
   5    0.1920000E+03  0.4908502E-01  0.3141441E+01
   6    0.3840000E+03  0.2454069E-01  0.3141208E+01
   7    0.7680000E+03  0.1227034E-01  0.3141208E+01
   8    0.1536000E+04  0.6127881E-02  0.3137475E+01
   9    0.3072000E+04  0.3049315E-02  0.3122499E+01
  10    0.6144000E+04  0.1582212E-02  0.3240371E+01
  11    0.1228800E+05  0.8457279E-03  0.3464102E+01
  12    0.2457600E+05  0.5980200E-03  0.4898980E+01
  13    0.4915200E+05  0.0000000E+00  0.0000000E+00
  14    0.9830400E+05  0.0000000E+00  0.0000000E+00
```

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **half1(r,b)**.
5. Обеспечить наглядный вывод результатов обеих расчетных формул в одну таблицу.
6. Включить **make**-файл псевдоцель вывода на один рисунок графиков, демонстрирующих приближение результатов к числу π с ростом $\mathbf{k}$.
7. Модифицировать программу для получения результатов с удвоенной точностью.

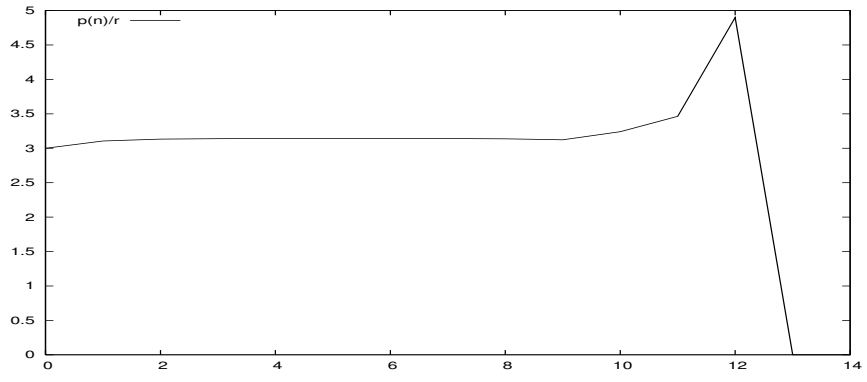

Рис. 3: График зависимости отношения $\dfrac{\mathbf{p(n)}}{\mathbf{r}}$ от количества удвоений числа сторон.

4.2 Табулирование функции $w(x) = \dfrac{1 + e^{-x} - 2e^{-x/2}}{x^2}$

Для двадцати значений аргумента $\mathbf{x} \in [\mathbf{5.000e-4}, \mathbf{1.500e-3}]$ равномерно распределенных по указанному промежутку вычислить таблицу значений функции:

$$\mathbf{w(x)} = \frac{\mathbf{1 + e^{-x} - 2e^{-x/2}}}{\mathbf{x^2}}$$

и построить ее график. Некто предложил для расчета программу:

```
      program tsfs2p02
      data ninp / 5 /
      data nres / 6 /
      open(unit=ninp,file='input')
      open(unit=nres,file='result')
      read(ninp,100) x0,xk
      read(ninp,101) n
      write(nres,1000) x0, xk, n
      write(nres, 1100)
      h=(xk-x0)/n
      do i=0,n
        x=x0+i*h
        r0=w0(x)
        write(nres,1101) i, x, r0
      enddo
      close(nres)
  100 format(e10.3)
  101 format(i10)
 1000 format(1x,'# x0=',e15.7,5x,'xk='e15.7,5x,'n=',i3)
 1100 format(1x,'# N ',7x,'x',12x,'w0(x)')
 1101 format(1x,i3,e15.7,e15.7)
      end
```

в которой табулирование велось непосредственно по приведенной выше формуле через вызов функции **w0**:

```
      function w0(x)
      w0= (1 + exp(-x) - 2 * exp(-x/2)) / x / x
      return
      end
```

Тестирование функции на промежутке [1.0,2.0] удовлетворило заказчика. В результате требуемого пропуска программа дала такой результат:

```
# x0=  0.5000000E-03     xk=  0.1500000E-02     n= 10
# N        x            w0(x)
  0  0.5000000E-03  0.0000000E+00
  1  0.6000000E-03  0.0000000E+00
  2  0.7000000E-03  0.2432842E+00
  3  0.8000000E-03  0.3725290E+00
  4  0.9000000E-03  0.2943439E+00
  5  0.1000000E-02  0.3576278E+00
  6  0.1100000E-02  0.2955602E+00
  7  0.1200000E-02  0.2483527E+00
  8  0.1300000E-02  0.2821522E+00
  9  0.1400000E-02  0.2432843E+00
 10  0.1500000E-02  0.2119276E+00
```

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **w1(x)**.
5. Обеспечить наглядный вывод результатов всех расчетных формул в одну таблицу.
6. Включить в **make**-файл вывод на один рисунок графиков всех трех способов расчета.
7. Модифицировать программу для получения результатов с удвоенной точностью.

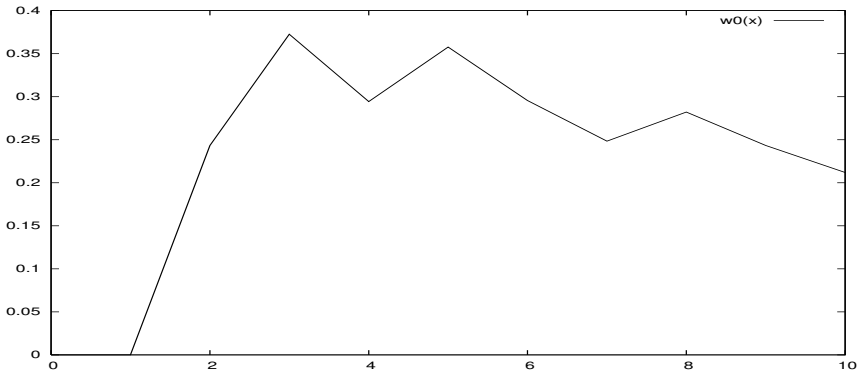

Рис. 4: График функции **w0(x)**

4.3 Отношение объемов.

Пользователь написал функцию **vsph(r)** расчета объема шара радиуса **r**:

```
      function vsph(r)                                !   Файл  vsph.for
      parameter (pi=3.14159265, c43=4.0/3.0)
      vsph=c43*pi*r*r*r
      end
```

и функцию **vcyl(r,h)** расчета объема кругового цилиндра высоты **h** с радиусом основания **r**:

```
      function vcyl(r,h)                              !   Файл vcyl.for
      parameter (pi=3.141592)
      vcyl=pi*r*r*h
      end
```

и применил их для поиска отношения объема шарового слоя, заключенного между сферами с радиусами **r** и **r+r*q** (q<1 – доля радиуса, посредством которой характеризуется толщина слоя) к объему цилиндра с площадью основания равной площади поверхности сферы радиуса **r** и высотой h=r*q, используя программу:

```
      program tsfs2p03                               !     Файл tsfs2p03.for
      data ninp / 5 /
      data nres / 6 /
      open(unit=ninp,file='input')
      open(unit=nres,file='result')
      pi=4.0*atan(1.0)
      write(nres,*) ' #  pi=',pi
      read(ninp,101) r, q0, q1
      write(nres,*) ' #   r=',r,'   q0=',q0, '   q1=',q1
      h=(q1-q0)/72
c      write(nres,*) ' #  h=',h
      write(nres,1000)
      do i=0,15
        q=q0+i*h
        rq=r*q
        res0=(vsph(r+rq)-vsph(r))/vcyl(2*r, rq)
        write(nres,1001) i,q, res0
      enddo
  101 format(e10.3)
 1000 format(1x,' #',2x,'i',12x,'q',10x,'res0')
 1001 format(1x,i5,2x,e15.7,e15.7,e15.7,e15.7)
      end
```

которая вводила значения трех параметров (**r** – радиуса сферы; **q0** и **q1** – концевых точек промежутка задания параметра **q**), вычисляла требуемое отношение для двадцати равноотстоящих по промежутку **[q0,q1]** значений **q**. Точность работы программы при **r=1.000** и $\mathbf{q} \in [\mathbf{4e-2}, \mathbf{4e-1}]$ заказчика устроила.

Расчет при **r=** и $\mathbf{q} \in [\mathbf{4e-8}, \mathbf{0.4e-7}]$ дал следующий результат

```
#  pi=   3.141593
#   r=   3.000000        q0=  4.0000000E-08    q1=  4.0000000E-07
#  i            q          res0
   0    0.4000000E-07  0.2248622E+01
   1    0.4500000E-07  0.1998775E+01
   2    0.5000000E-07  0.1798898E+01
   3    0.5500000E-07  0.1635362E+01
   4    0.6000000E-07  0.1499082E+01
   5    0.6500000E-07  0.1383768E+01
   6    0.7000000E-07  0.1284927E+01
   7    0.7500000E-07  0.1199265E+01
   8    0.8000000E-07  0.1124311E+01
   9    0.8500000E-07  0.1058175E+01
  10    0.9000000E-07  0.9993877E+00
  11    0.9500000E-07  0.9467884E+00
  12    0.1000000E-06  0.8994489E+00
  13    0.1050000E-06  0.8566180E+00
  14    0.1100000E-06  0.8176808E+00
  15    0.1150000E-06  0.7821296E+00
```

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **ratio1(q)**.
5. Обеспечить наглядный вывод результатов всех расчетных формул в одну таблицу.
6. Включить в **make**-файл вывод на один рисунок графиков всех трех способов расчета.
7. Модифицировать программу для получения результатов с удвоенной точностью.

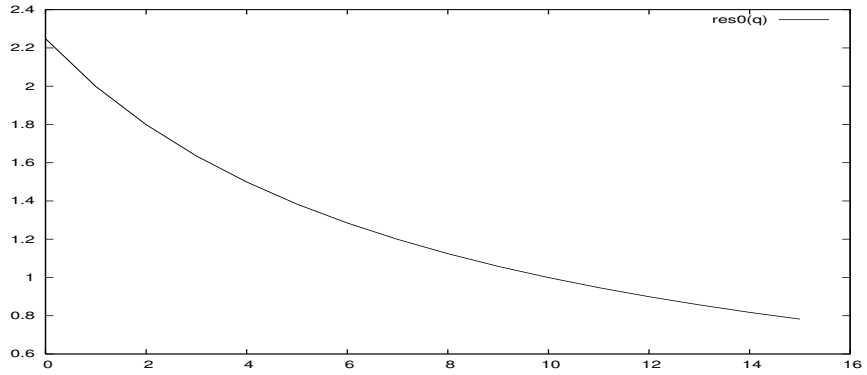

Рис. 5: График искомого отношения при $\mathbf{r{=}3}$ и $\mathbf{q} \in [\mathbf{4e-8}, \mathbf{4e-7}]$

4.4 Косинус угла сферического треугольника.

Косинус угла сферического равностороннего треугольника со стороной, измеряемой в радианной мере дугой φ, выражается через нее формулой:

$$\cos(\mathbf{A}) = \frac{\cos(\varphi) - \cos^{\mathbf{2}}(\varphi)}{\sin^{\mathbf{2}}(\varphi)}$$

Некто решил проверить, что по мере уменьшения длины стороны φ величина $\cos(\mathbf{A})$ будет приближаться к **0,5**. Программа

```
      program tsfs2p04
      data ninp / 5 /, nres / 6 /
      real xx (100)
      open(unit=ninp,file='input')
      open(unit=nres,file='result')
      pi=4*atan(1.0)
      cpi3=cos(pi/3)
      read(ninp,100) n
      read(ninp,101) (xx(i),i=1,n)
      write(nres,1000) n
      write(nres, 1100)
      do i=1,n
        x=xx(i)
        resm0=cosa0(x)
        aer0=abs(resm0-cpi3)
        rer0=aer0/cpi3
        write(nres,1101) i, x, resm0, aer0, rer0
      enddo
      close(nres)
  100 format(i10)
  101 format(e10.3)
 1000 format(1x,'#  n=',i3)
 1100 format(1x,'#N ',5x,'x',13x,'cosa0',9x,'aer',7x,'rer')
 1101 format(i3,e14.7,e16.7,e10.2,e10.2)
      end
```

для **n** вводимых значений аргумента $\varphi \in [\mathbf{0.5}, \mathbf{1e-9}]$ вычислила по приведенной формуле (посредством функции **cosa0**)

```
      function cosa0(x)
      cosa0= (cos(x)-cos(x)**2)/sin(x)**2
      end
```

таблицу значений искомого косинуса и их абсолютные и относительные погрешности относительно правильного значения.
В результате требуемого пропуска программа дала такой результат:

```
#  n= 18
#N      x              cosa0          aer       rer
 1 0.5000000E+00   0.4674003E+00  0.33E-01  0.65E-01
 2 0.5000000E-01   0.4996850E+00  0.31E-03  0.63E-03
 3 0.2500000E-01   0.4999623E+00  0.38E-04  0.75E-04
 4 0.1000000E-01   0.5001868E+00  0.19E-03  0.37E-03
 5 0.5000000E-02   0.5013633E+00  0.14E-02  0.27E-02
 6 0.1000000E-02   0.4536744E+00  0.46E-01  0.93E-01
 7 0.5000000E-03   0.4536743E+00  0.46E-01  0.93E-01
 8 0.4500000E-03   0.6773758E+00  0.18E+00  0.35E+00
 9 0.4000000E-03   0.2450581E+00  0.25E+00  0.51E+00
10 0.3500000E-03   0.4731372E+00  0.27E-01  0.54E-01
11 0.3000000E-03   0.8245475E+00  0.32E+00  0.65E+00
12 0.2500000E-03   0.1407348E+01  0.91E+00  0.18E+01
13 0.2000000E-03  -0.5000000E+00  0.10E+01  0.20E+01
14 0.1000000E-03  -0.5000000E+00  0.10E+01  0.20E+01
15 0.1000000E-06  -0.5000015E+00  0.10E+01  0.20E+01
16 0.1000000E-07  -0.4998172E+00  0.10E+01  0.20E+01
17 0.1000000E-08  -0.4878910E+00  0.99E+00  0.20E+01
18 0.1000000E-09   0.0000000E+00  0.50E+00  0.10E+01
```

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен (два–три способа).
4. Написать соответствующие новым схемам расчета функции **cosa1(x)** и **cosa2(x)**.
5. Обеспечить наглядный вывод относительных погрешностей расчета по всем формулам в одну таблицу.
6. Включить в **make**-файл вывод на один рисунок графиков всех трех способов расчета.
7. Модифицировать программу для получения результатов с удвоенной точностью.

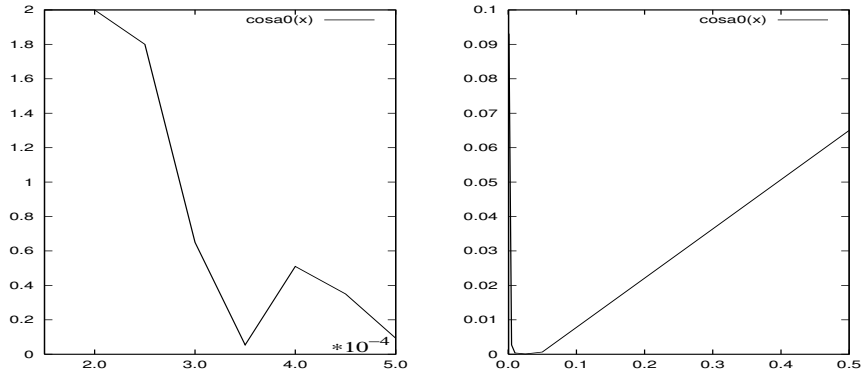

Рис. 6: Относительная погрешность работы **cosa0(x)**

4.5 Расчет отношения $r(x) = \dfrac{1.1 - \sqrt{1.21 - x^2}}{x \cdot (2.2 - \sqrt{4.84 - x})}$ при $\mathbf{x < 1}$

В некоторой задаче потребовался расчет отношения **r**(**x**). Были составлены две подпрограммы функции **r0**(**x**) и **r1**(**x**):

```
      function r0(x)
      r0= (1.1-sqrt(1.21-x*x))/(2.2-sqrt(4.84-x))/x
      end
      function r1(x)
      r1= 0.5*(1-sqrt(1-x*x/1.21))/(1-sqrt(1-x/4.84))/x
      end
```

Первая вела расчет непосредственно по формуле из заголовка. Вторая по чуть измененной, но аналитически тождественной. Именно, для уменьшения влияния погрешностей округления из под знака квадратного корня были вынесены константы **1,21** и **4,84** (соответственно в числителе и знасменателе) так, что оказалось возможным ранее приближенные значения **1,1**, **2,2** заменить на точную единицу. Тестирование обеих функций проводилось программой

```
      program tsfs2p05
      data ninp / 5 /
      data nres / 6 /
      open(unit=ninp,file='input')
      open(unit=nres,file='result')
      read(ninp,100) x0,xk
      read(ninp,101) n
      write(nres,1000) x0, xk, n
      write(nres, 1100)
      h=(xk-x0)/n
      do i=0,n
        x=x0+i*h
         res0=r0(x)
         res1=r1(x)
        write(nres,1101) i, x, res0, res1
      enddo
      close(nres)
  100 format(e10.3)
  101 format(i10)
 1000 format(1x,'# x0=',e15.7,5x,'xk='e15.7,5x,'n=',i3)
 1100 format(1x,'# N ',7x,'x',12x,'r0(x)',12x,'r1')
 1101 format(1x,i3,e15.7,e15.7,e15.7)
      end
```

на значениях аргумента $\mathbf{x} \in [\mathbf{0.49}, \mathbf{0.59}]$ и дало практически совпадающие результаты. Однако, тестирование функции на требуемом рабочем диапазоне $[\mathbf{0.00049}, \mathbf{0.00059}]$ привело к следующему:

```
# x0=  0.4900000E-03     xk=  0.5900000E-03     n= 10
# N        x            r0(x)            r1
  0  0.4900000E-03  0.2185028E+01  0.2400960E+01
  1  0.5000000E-03  0.2096436E+01  0.2306805E+01
  2  0.5100000E-03  0.2017268E+01  0.2218082E+01
  3  0.5200000E-03  0.1938586E+01  0.2132014E+01
  4  0.5300000E-03  0.1868111E+01  0.2053093E+01
  5  0.5400000E-03  0.1797914E+01  0.1978474E+01
  6  0.5500000E-03  0.1734906E+01  0.1905851E+01
  7  0.5600000E-03  0.1672017E+01  0.1839047E+01
  8  0.5700000E-03  0.1615457E+01  0.2663541E+01
  9  0.5800000E-03  0.1558895E+01  0.2573340E+01
 10  0.5900000E-03  0.1507932E+01  0.2485213E+01
```

Как видно, совпадающих результатов нет. **Какой же из функций верить?**

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен (два–три способа).
4. Написать соответствующие новым схемам расчета функции **r2(x)** и **r3(x)**.
5. Обеспечить наглядный вывод результатов расчета по всем четырем формулам в одну таблицу.
6. Включить в **make**-файл вывод на один рисунок четырех, соответствующих таблице графиков.
7. Модифицировать программу для получения результатов с удвоенной точностью.

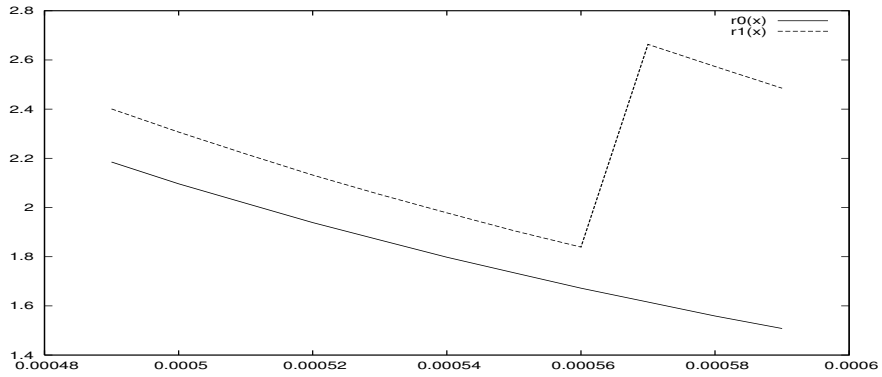

Рис. 7: Графики r(x), полученные r0(x) и r1(x)

4.6 Табулирование отношения $\dfrac{\tan x - x}{x - \sin x}$ при $x \ll 1$.

Нужна таблица значений при $x = exp(-t)$ и $t = 20.4(0.05)21.4$. Были составлены две подпрограммы-функции **d0**(**x**) и **d1**(**x**):

```
c                                                    файл d0.for
      function d0(x)
      d0= (tan(x)-x)/(x-sin(x))
      end

c                                                    файл d1.for
      function d1(x)
      d1= (tan(x)/x-1)/(1-sin(x)/x)
      end
```

Первая вела расчет непосредственно по формуле из заголовка. Вторая по чуть измененной, но аналитически тождественной. Именно, для уменьшения влияния погрешностей округления и в числителе, и в знаменателе было вынесен за скобки аргумент **x** так, что оказалось возможным одно из слагаемых заменить на точную единицу. Тестирование обеих функций проводилось программой

```
      program tsfs2p06
      data ninp / 5 /
      data nres / 6 /
      open(unit=ninp,file='input')
      open(unit=nres,file='result')
      read(ninp,100) t0,tk
      read(ninp,101) n
      write(nres,1000) t0, tk, n
      write(nres, 1100)
      h=(tk-t0)/n
      do i=1,n
         t=t0+h*(i-1)
         x=exp(-t)
         r0=d0(x)
         r1=d1(x)
         write(nres,1101) i, t, r0, r1
      enddo
      close(nres)
  100 format(e10.3)
  101 format(i10)
 1000 format(1x,'# t0=',e15.7,5x,'tk='e15.7,5x,'n=',i3)
 1100 format(1x,'# N ',7x,'t',9x,'d0(exp(-t))',4x,'d1(exp(-t))')
 1101 format(1x,i3,e15.7,e15.7,e15.7)
      end
```

и на значениях аргумента $\mathbf{t} \in [\mathbf{1.0}, \mathbf{2.0}]$ дало практически совпадающие результаты. Однако, тестирование функции на требуемом рабочем диапазоне $[\mathbf{20.4}, \mathbf{21.4}]$ привело к следующему:

```
# t0=  0.2040000E+02     tk=  0.2140000E+02     n= 10
# N        t          d0(exp(-t))    d1(exp(-t))
  1  0.2040000E+02  0.2250000E+01  0.2400000E+01
  2  0.2050000E+02  0.2000000E+01  0.2000000E+01
  3  0.2060000E+02  0.2500000E+01  0.2666667E+01
  4  0.2070000E+02  0.1500000E+01  0.1500000E+01
  5  0.2080000E+02  0.1666667E+01  0.2000000E+01
  6  0.2090000E+02  0.2000000E+01  0.2000000E+01
  7  0.2100000E+02  0.3000000E+01  0.4000000E+01
  8  0.2110000E+02  0.2000000E+01  0.2000000E+01
  9  0.2120000E+02  0.2000000E+01  0.2000000E+01
 10  0.2130000E+02  0.1000000E+01  0.1000000E+01
```

Некоторая странность результатов налицо. **Какой же из функций верить?**

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен (два–три способа).
4. Написать соответствующие новым схемам расчета функции **d2(x)** и **d3(x)**.
5. Обеспечить наглядный вывод результатов расчета по всем четырем формулам в одну таблицу.
6. Включить в **make**-файл вывод на один рисунок четырех, соответствующих таблице графиков.
7. Модифицировать программу для получения результатов с удвоенной точностью.

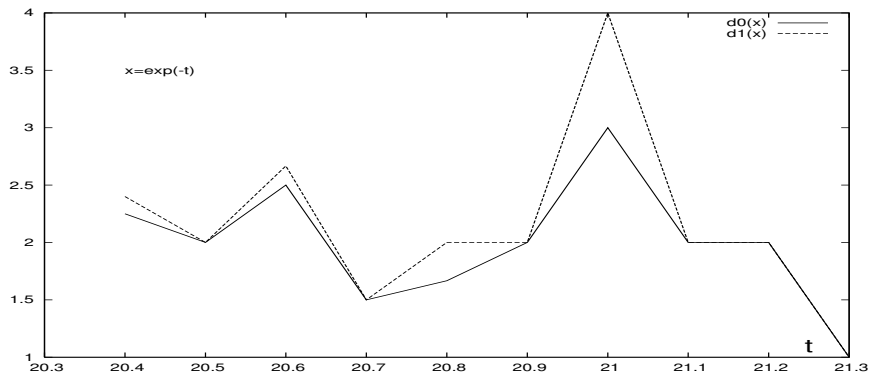

Рис. 8: Графики искомого отношения, полученные функциями d0(x) и d1(x)

4.7 Табулирование функции $(1-x)tg\frac{\pi x}{2}$ при $x \to 1$.

Нужна таблица значений при $x = 1 - k \cdot 10^{-6}$ и $k = 1(1)10$. Была составлена подпрограмма-функция $\mathbf{q0(x)}$:

```
c                                                    файл d0.for
      function q0(x)
      pi2=2*atan(1e0)
      q0= (1-x)*(tan(pi2*x))
      end
```

которая вела расчет непосредственно по формуле из заголовка. Тестирование функции проводилось программой

```
      program tsfs2p07
      data ninp / 5 /
      data nres / 6 /
      open(unit=ninp,file='input')
      open(unit=nres,file='result')
      write(nres,'('' # 2/pi='',e15.7)') 2/(4*atan(1e0))
      read(ninp,100) t0,tk
      read(ninp,101) n
      write(nres,1000) t0, tk, n
      write(nres, 1100)
      h=(tk-t0)/n
      do i=0,n
         t=(t0+h*i)*1e-6
         x=1-t
         r0=q0(x)
         write(nres,1101) i, t, x, r0
      enddo
      close(nres)
  100 format(e10.3)
  101 format(i10)
 1000 format(1x,'# t0=',e15.7,5x,'tk='e15.7,5x,'n=',i3)
 1100 format(1x,'# N ',7x,'t',14x,'x',13x,'r0')
 1101 format(1x,i3,e15.7,e15.7,e15.7)
      end
```

При $\mathbf{t = 0,1(0,05)0.6}$, что соответствует $\mathbf{x = 0,9(-0,05)0.4}$) тестирование дало результаты, устраивающие заказчика. В частности, при $\mathbf{t = 0,5}$ аргумент $\mathbf{x}$ тоже равен половине, как и $(\mathbf{1 - x})$, а $\mathbf{tg\frac{\pi}{4} = 1}$. Так что семь цифр мантиссы результата на одинарной точности верны. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{10^{-6}}, \mathbf{10^{-5}}]$ привело к следующему:

```
# 2/pi=  0.6366197E+00
# t0=  0.1000000E+00     tk=  0.1100000E+01     n= 10
# N        t              x             r0
  0  0.1000000E-06  0.9999999E+00  0.6122490E+00
  1  0.2000000E-06  0.9999998E+00  0.9183736E+00
  2  0.3000000E-06  0.9999997E+00  0.6880755E+00
  3  0.4000000E-06  0.9999996E+00  0.7553975E+00
  4  0.5000000E-06  0.9999995E+00  0.7100605E+00
  5  0.6000000E-06  0.9999994E+00  0.6550228E+00
  6  0.7000000E-06  0.9999993E+00  0.6949816E+00
  7  0.8000000E-06  0.9999992E+00  0.6747413E+00
  8  0.9000000E-06  0.9999991E+00  0.6446998E+00
  9  0.1000000E-05  0.9999990E+00  0.6728238E+00
 10  0.1100000E-05  0.9999989E+00  0.6601472E+00
```

На первый взгляд результаты кажутся правдоподобными. Более внимательный их анализ заставит насторожиться: именно, странна их немонотонность. Насколько они верны?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **q1**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод на один рисунок обоих графиков, соответствующих таблице.
7. Модифицировать программу для получения результатов с удвоенной точностью.

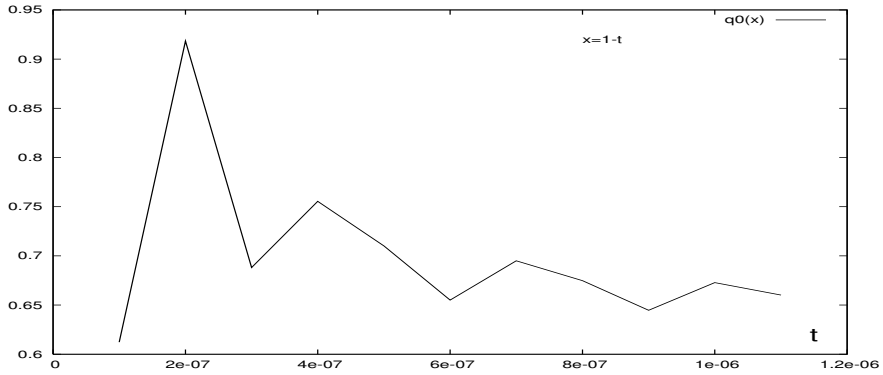

Рис. 9: График функции, табулированной на основе вызовов q0(x)

4.8 Табулирование функции $f(x) = \dfrac{p - \sqrt{p^2 - x^7}}{x^7}$

Для двадцати значений аргумента $\mathbf{x} \in [\mathbf{0.1165}, \mathbf{0.1365}]$ равномерно распределенных по указанному промежутку и вводимого параметра p вычислить таблицу значений функции:

$$f(x) = \frac{p - \sqrt{p^2 - x^7}}{x^7}$$

и построить ее график. Некто предложил для расчета программу:

```
      program tsfs2p08                        !    Файл  tsfs2p08.for
      data ninp / 5 /
      data nres / 6 /
      data a, b, n / 0.1165, 0.1365, 15 /
      open(unit=ninp,file='input')
      open(unit=nres,file='result')
      read(ninp,101) p
      write(nres,*) ' #   p=',p
      write(nres,1000)
      h=(b-a)/n
      do i=1,n+1
        x=a+(i-1)*h
        r0=f0(p,x)
        r1=f1(p,x)
        write(nres,1001) i, x, r0, r1
      enddo
c                                        Форматы ввода-вывода:
  101 format(e10.3)
 1000 format(1x,' #',2x,'i',9x,'x',12x,'f0(x)',10x,'f1(x)')
 1001 format(1x,i5,2x,e15.7,e15.7,e15.7)
      end
```

в которой табулирование велось двумя способами: через вызовы функций **f0** и **f1**:

```
      function f0(p,x)                        !      Файл  f0.for
      x7=x**7
      p2=p*p
      f0=(p-sqrt(p2-x7))/x7
      end

      function f1(p,x)                        !     Файл   f1.for
      f1=(p-sqrt(p*p-x**7))/x**7
      end
```

В результате пропуска при $\mathbf{p = 1.3}$ дала такой результат:

```
#   p=  1.29999995
#  i         x             f0(x)          f1(x)
   1    0.1165000E+00  0.4092877E+00  0.4092877E+00
   2    0.1178333E+00  0.3779489E+00  0.3779488E+00
   3    0.1191667E+00  0.3493226E+00  0.3493225E+00
   4    0.1205000E+00  0.3231475E+00  0.3231475E+00
   5    0.1218333E+00  0.2991901E+00  0.2991901E+00
   6    0.1231667E+00  0.5544825E+00  0.2772412E+00
   7    0.1245000E+00  0.5142267E+00  0.2571134E+00
   8    0.1258333E+00  0.4772767E+00  0.4772767E+00
   9    0.1271667E+00  0.4433301E+00  0.4433302E+00
  10    0.1285000E+00  0.4121148E+00  0.4121148E+00
  11    0.1298333E+00  0.3833862E+00  0.3833862E+00
  12    0.1311667E+00  0.3569240E+00  0.3569240E+00
  13    0.1325000E+00  0.3325287E+00  0.3325287E+00
  14    0.1338333E+00  0.4650303E+00  0.3100202E+00
  15    0.1351667E+00  0.4338543E+00  0.4338543E+00
  16    0.1365000E+00  0.4050446E+00  0.4050445E+00
```

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **f2(p,b)**.
5. Обеспечить наглядный вывод результатов всех расчетных формул в одну таблицу.
6. Включить в **make**-файл вывод на один рисунок графиков всех трех способов расчета.
7. Модифицировать программу для получения результатов с удвоенной точностью.

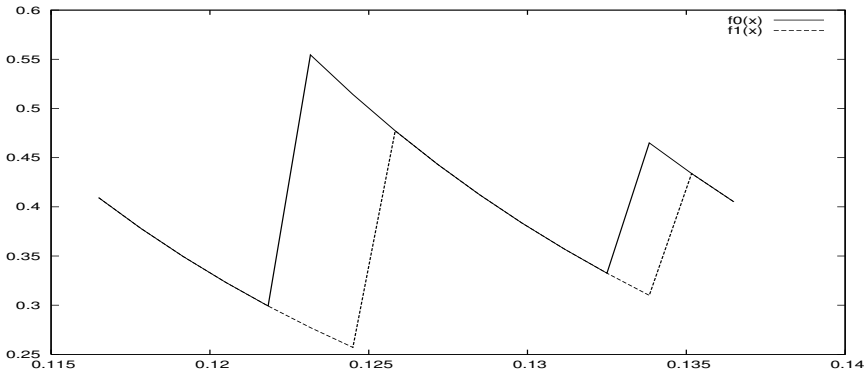

Рис. 10: **p=1.3** Графики функций **f0(x)** и **f1(x)**.

4.9 Табулирование отношения $\dfrac{1-x^{1/2}}{1-x^{1/3}}$ при $x \to 1$

Нужна таблица значений при $\mathbf{x = exp(-t^3)}$ и **t=0.0045(0.05)0.0095**.
Была составлена подпрограмма-функция **w0(x)**:

```
c                                                    файл w0.for
      function w0(x)
      w0=(1-sqrt(x))/(1-exp(alog(x)/3))
      end
```

которая вела расчет непосредственно по формуле из заголовка.
Тестирование функции проводилось программой

```
      program tsfs2p07
      data ninp / 5 /
      data nres / 6 /
      open(unit=ninp,file='input')
      open(unit=nres,file='result')
      read(ninp,100) t0,tk
      read(ninp,101) n
      write(nres,1000) t0, tk, n
      write(nres, 1100)
      h=(tk-t0)/n
      do i=0,n
         t=t0+i*h
         x=exp(-t*t*t)
         r0=w0(x)
         write(nres,1101) i, t, x, r0
      enddo
      close(nres)
  100 format(e10.3)
  101 format(i10)
 1000 format(1x,'# t0=',e15.7,5x,'tk='e15.7,5x,'n=',i3)
 1100 format(1x,'# N ',7x,'t',14x,'x',13x,'r0')
 1101 format(1x,i3,e15.7,e15.7,e15.7)
      end
```

При **t=1(0,1)2** тестирование дало результаты, устраивающие заказчика. Однако, тестирование функции на требуемом рабочем диапазоне при **t=0,0045(0,0005)0,0095** привело к следующему:

```
# t0=  0.4500000E-02     tk=  0.9500000E-02     n= 10
# N        t              x             r0
  0  0.4500000E-02  0.9999999E+00  0.1000000E+01
  1  0.5000000E-02  0.9999999E+00  0.1000000E+01
  2  0.5500000E-02  0.9999998E+00  0.2000000E+01
  3  0.6000000E-02  0.9999998E+00  0.2000000E+01
  4  0.6500000E-02  0.9999997E+00  0.1500000E+01
  5  0.7000000E-02  0.9999996E+00  0.1500000E+01
  6  0.7499999E-02  0.9999996E+00  0.2000000E+01
  7  0.7999999E-02  0.9999995E+00  0.1666667E+01
  8  0.8500000E-02  0.9999994E+00  0.1666667E+01
  9  0.9000000E-02  0.9999993E+00  0.1500000E+01
 10  0.9500000E-02  0.9999992E+00  0.1400000E+01
```

Насколько верны эти результаты?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **w1**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод на один рисунок обоих графиков, соответствующих таблице.
7. Модифицировать программу для получения результатов с удвоенной точностью.

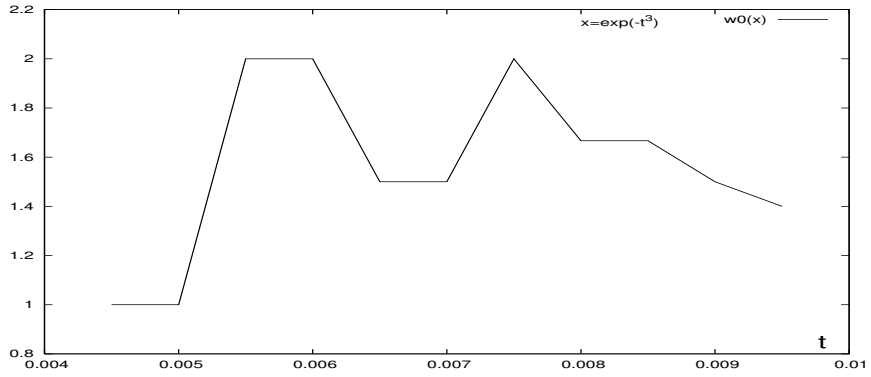

Рис. 11: График функции, табулированной на основе вызовов w0(x)

4.10 Табулирование отношения $\frac{\sin^2 x}{1+\cos^3 x}$ при $x \to \pi$

Нужна таблица значений при $\mathbf{x} = \pi + \mathbf{t} \cdot \mathbf{2} \cdot \mathbf{10^{-3}}$ и $\mathbf{t} = -\mathbf{1}(\mathbf{0},\mathbf{1})\mathbf{1}$.
Была составлена подпрограмма-функция **rsi0**(**x**):

```
c                                                    файл rsi0.for
      function rsi0(x)
      rsi0=sin(x)**2/(1+cos(x)**3)
      end
```

которая вела расчет непосредственно по формуле из заголовка.
Тестирование функции проводилось программой

```
      program tsfs2p10
      data ninp / 5 /
      data nres / 6 /
      open(unit=ninp,file='input')
      open(unit=nres,file='result')
      pi=4*atan(1e0)
      read(ninp,100) t0,tk
      read(ninp,101) n
      write(nres,1000) t0, tk, n
      write(nres, 1100)
      h=(tk-t0)/n
      do i=0,n
         t=t0+i*h
         x=pi+2e-3*t
         r0=rsi0(x)
         write(nres,1101) i, t, x, r0
      enddo
      close(nres)
  100 format(e10.3)
  101 format(i10)
 1000 format(1x,'# t0=',e15.7,5x,'tk='e15.7,5x,'n=',i3)
 1100 format(1x,'# N ',7x,'t',14x,'x',13x,'r0')
 1101 format(1x,i3,e15.7,e15.7,e15.7)
      end
```

При $\mathbf{t} = -\mathbf{1000}(\mathbf{200})\mathbf{1000}$ тестирование дало результаты, устраивающие заказчика.
Однако, тестирование функции на требуемом рабочем диапазоне при $\mathbf{t} = [-\mathbf{1},(\mathbf{0.2})\mathbf{1}]$ привело к следующему:

```
# t0= -0.1000000E+01     tk=  0.1000000E+01     n= 10
# N        t              x             r0
  0 -0.1000000E+01  0.3139593E+01  0.6579345E+00
  1 -0.8000000E+00  0.3139993E+01  0.6816897E+00
  2 -0.6000000E+00  0.3140393E+01  0.6709471E+00
  3 -0.4000000E+00  0.3140793E+01  0.7154825E+00
  4 -0.2000000E+00  0.3141193E+01  0.8946908E+00
  5  0.1490116E-07  0.3141593E+01      +Infinity
  6  0.2000000E+00  0.3141993E+01  0.8954731E+00
  7  0.4000000E+00  0.3142393E+01  0.7157953E+00
  8  0.6000000E+00  0.3142793E+01  0.6711426E+00
  9  0.8000000E+00  0.3143193E+01  0.6818387E+00
 10  0.1000000E+01  0.3143593E+01  0.6580495E+00
```

Насколько верны эти результаты?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **rsi1**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод на один рисунок обоих графиков, соответствующих таблице.
7. Модифицировать программу для получения результатов с удвоенной точностью.

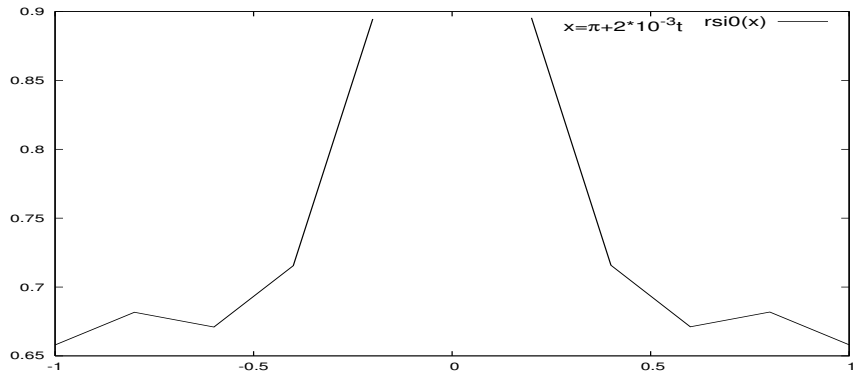

Рис. 12: График функции, табулированной на основе вызовов rsi0(x)

4.11 Проверка формулы $\frac{a^2-b^2}{a-b}=a+b$

Школьник решил проверить численно на компьютере правильность алгебраической формулы из заголовка. Программа вводила **b** и вычисляла **a**, прибавляя к содержимому переменной **b** некоторое уменьшающееся приращение **x**, после чего расчет левой части проверяемой формулы выполнялся тремя способами при запомненных в переменных **a2** и **b2** значениях a^2 и b^2 соответственно:

1. **(a2-b2)/(a-b)**
2. **(a2-b2)/ x**
3. **ab2:=a2-b2; ab2/x**

```
      program tsfs2p11
      data ninp / 5 /
      data nres / 6 /
      open(unit=ninp,file='input')
      open(unit=nres,file='result')
      read(ninp,100) b
      write(nres,1000) b
      write(nres, 1100)
      x=b
      b2=b*b
      do while (a.ne.b)
        a=b+x
        a2=a*a
        ab2=a2-b2;
        r0=(a2-b2)/(a-b)
        r1=(a2-b2)/x
        r2=ab2/x
        write(nres,1101) x, r0, r1, r2
        x=x/10
      enddo
      close(nres)
  100 format(e10.3)
 1000 format(1x,'# b=',e15.7,10x,' a=b+x')
 1100 format(1x,'#',16x/1x,'#',16x,'a2=a*a b2=b*b',18x,'ab2=a2-b2'/
     > 1x,'#',7x,'x',8x,'(a2-b2)/(a-b)',4x,'(a2-b2)/x',6x,'  ab2/x  ')
 1101 format(1x,e15.7,e15.7,e15.7,e15.7)
      end
```

Начальное значения аргумента **x** полагалось равным **b**, каждое следующее вдесятеро меньше предыдущего. Уменьшение прекращалось, как только значение переменной **a** “с точки зрения ЭВМ” в точности оказывалось равной значению **b**.
Помогите школьнику понять получаемые результаты:

```
# b=  0.1100000E+01           a=b+x
#
#                a2=a*a b2=b*b                   ab2=a2-b2
#       x        (a2-b2)/(a-b)    (a2-b2)/x        ab2/x
  0.1100000E+01  0.3300000E+01  0.3300000E+01  0.3300000E+01
  0.1100000E+00  0.2310000E+01  0.2310001E+01  0.2310001E+01
  0.1100000E-01  0.2211000E+01  0.2211007E+01  0.2211007E+01
  0.1100000E-02  0.2201149E+01  0.2201037E+01  0.2201037E+01
  0.1100000E-03  0.2200433E+01  0.2201037E+01  0.2201037E+01
  0.1100000E-04  0.2206522E+01  0.2199953E+01  0.2199953E+01
  0.1100000E-05  0.2222222E+01  0.2167442E+01  0.2167442E+01
  0.1100000E-06  0.2000000E+01  0.2167442E+01  0.2167442E+01
  0.1100000E-07             NaN  0.0000000E+00  0.0000000E+00
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **r3**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод на один рисунок трех графиков, соответствующих таблице.
7. Модифицировать программу для получения результатов с удвоенной точностью.

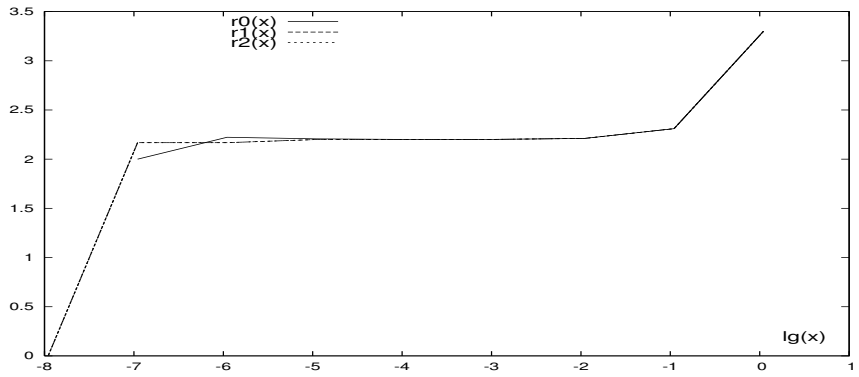

Рис. 13: Графики зависимости частных **r0(x)**, **r1(x)**, **r2(x)**

4.12 Табулирование функции $\dfrac{\sin(x)}{\sqrt{1+tg(x)}-\sqrt{1-tg(x)}}$

В некоторой задаче потребовалась табулировать указанную функцию для $\mathbf{x} = \mathbf{e^{-t}}$ при $t \in [15.5, 16.5]$. Была составлена подпрограмма функции $\mathbf{q0}(\mathbf{t})$:

```
      function q0(x)                          !      Файл  q0.for
      q0=sin(x)/(sqrt(1+tan(x))-sqrt(1-tan(x)))
      end
```

которая вела расчет непосредственно по формуле из заголовка. Тестирование функции проводилось программой

```
      program tsfs2p12                        !    Файл  tsfs2p12.for
      data ninp / 5 /
      data nres / 6 /
      open(unit=ninp,file='input')
      open(unit=nres,file='result')
      read(ninp,100) t0, tn
      read(ninp,101) n
      write(nres,*) ' #   t0=',t0,'  tn=',tn,'  n=',n
      write(nres,1000)
      ht=(tn-t0)/n
      do i=0,n
        t=t0+i*ht
        x=exp(-t)
        r0=q0(x)
        write(nres,1001) i, t, x, r0
      enddo
c                                        Форматы ввода-вывода:
  100 format(e15.7)
  101 format(i15)
 1000 format(1x,' #',2x,'i',10x,'t',14x,'x',13x,'r0')
 1001 format(1x,i5,2x,e15.7,e15.7,e15.7)
      end
```

на значениях аргумента $\mathbf{x} \in [\pi/\mathbf{6}, \pi/\mathbf{4}]$ дало результаты, удовлетворяющие заказчика. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{15.5}, \mathbf{16.5}]$ привело к следующему:

```
#   t0=  15.5  tn=  16.5  n= 10
#  i          t              x             r0
   0    0.1550000E+02  0.1855391E-06  0.7782075E+00
   1    0.1560000E+02  0.1678827E-06  0.9388680E+00
   2    0.1570000E+02  0.1519066E-06  0.8495234E+00
   3    0.1580000E+02  0.1374507E-06  0.7686803E+00
   4    0.1590000E+02  0.1243706E-06  0.6955311E+00
   5    0.1600000E+02  0.1125352E-06  0.1888027E+01
   6    0.1610000E+02  0.1018260E-06  0.1708357E+01
   7    0.1620000E+02  0.9213594E-07  0.1545785E+01
   8    0.1630000E+02  0.8336817E-07  0.1398686E+01
   9    0.1640000E+02  0.7543461E-07  0.1265583E+01
  10    0.1650000E+02  0.6825604E-07  0.1145146E+01
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **f1**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод на один рисунок трех графиков, соответствующих таблице.
7. Модифицировать программу для получения результатов с удвоенной точностью.

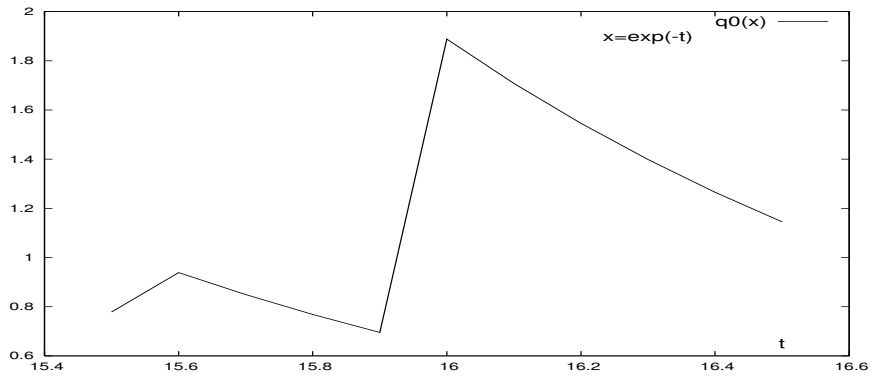

Рис. 14: График функции, табулированной на основе вызовов f0(x)

4.13 Табулирование функции $(1+x)^{\frac{1}{x}}$

В некоторой задаче потребовалась табулировать указанную функцию для $\mathbf{x} = \mathbf{10^{-t}}$ при $t = 15(0.1)16]$. Были составлены две подпрограммы-функции **u0**(**x**) и **u1**(**x**):

```
      function u0(x)                          !      Файл  u0.for
      u0=(1+x)**(1.0/x)
      end

      function u1(x)                            ! файл u1.for
      x1=1+x
      u1=x1**(1.0/x)
      end
```

u0(**x**) вела расчет непосредственно по формуле из заголовка, а **u1**(**x**), в отличие от **u0**(**t**), значение основания предварительно записывала в рабочую переменную, возводя в степень содержимое последней. Тестирование функции проводилось программой

```
      program tsfs2p13                        !    Файл  tsfs2p13.for
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100) t0, tn
      read(ninp,101) n
      write(nres,*)   ' #   t0=',t0,'  tn=',tn,'  n=',n
      write(nres,1000)
      ht=(tn-t0)/n
      do i=0,n
        t=t0+i*ht
        x=10**(-t)
        r0=u0(x)
        r1=u1(x)
        write(nres,1001) i, t, x, r0, r1
      enddo
  100 format(e15.7)
  101 format(i15)
 1000 format(1x,' #',2x,'i',10x,'t',13x,'x',14x,'r0',13x,'r1')
 1001 format(1x,i5,2x,e15.7,e15.7,e15.7,e15.7)
      end
```

на значениях аргумента $\mathbf{t} = [\mathbf{7}, \mathbf{11}]$ дало результаты, удовлетворяющие заказчика и совпадающие численно с точностью до семи значащих цифр на одинарной точности с аналитическим значением второго замечательного предела. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{15}, \mathbf{16}]$ привело к следующему:

```
#   t0=   15.00000       tn=   16.00000       n=          10
#  i          t             x             r0            r1
    0    0.1500000E+02  0.1000000E-14  0.3035035E+01  0.1000000E+01
    1    0.1510000E+02  0.7943275E-15  0.3059194E+01  0.1000000E+01
    2    0.1520000E+02  0.6309576E-15  0.2874131E+01  0.1000000E+01
    3    0.1530000E+02  0.5011870E-15  0.2425590E+01  0.1000000E+01
    4    0.1540000E+02  0.3981075E-15  0.3051095E+01  0.1000000E+01
    5    0.1550000E+02  0.3162278E-15  0.2018121E+01  0.1000000E+01
    6    0.1560000E+02  0.2511884E-15  0.2420505E+01  0.1000000E+01
    7    0.1570000E+02  0.1995263E-15  0.3043045E+01  0.1000000E+01
    8    0.1580000E+02  0.1584892E-15  0.4059287E+01  0.1000000E+01
    9    0.1590000E+02  0.1258927E-15  0.5834342E+01  0.1000000E+01
   10    0.1600000E+02  0.1000000E-15  0.1000000E+01  0.1000000E+01
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **u2(x)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **u0(x)**, **u1(x)**, **u2(x)**.
7. Модифицировать программу для получения результатов с удвоенной точностью.

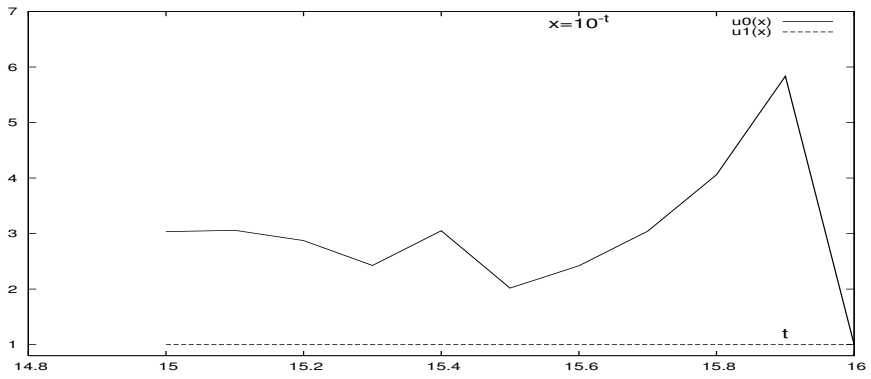

Рис. 15: Графики функций u0(x) и u1(x) при x=exp(-t) (по оси абсцисс отложено t).

4.14 Расчет константы из формулы Стирлинга

Школьник, прочитав о том, что формула Стирлинга

$$\ln(\mathbf{n}!) = \mathbf{C} + (\mathbf{n} + \frac{\mathbf{1}}{\mathbf{2}}) \cdot \ln(\mathbf{n}) - \mathbf{n} + \frac{\theta}{\mathbf{12n}}, \quad (\mathbf{0} < \theta < \mathbf{1}),$$

пригодна для практических вычислений, решил вычислить входящую в нее константу **C**, вычтя из левой части второе и третье слагаемые правой, полагая, что при достаточно больших **n** величина константы вычислится с неплохой точностью. Например, при $n = 10^5$ погрешность приведенной формулы Стирлинга не хуже чем $\mathbf{10^{-5}}$. Была составлена функция:

```
      function c0(n)                          !      Файл  c0.for
      dn=n
      fln=0.0
      do i=n,2,-1
        di=i
        fln=fln+alog(di)
      enddo
      s23=(dn+0.5)*alog(dn)-dn
      c0=fln-s23
      end
```

Тестирование ее посредством программы

```
      program tsfs2p14              !    Файл  tsfs2p14.for
      integer nn(14)
      data ninp / 5 /, nres / 6 /
      pi=4*atan(1.0)
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100) n
      write(nres, *) ' #   n=',n,'      C=',0.5*alog(2*pi)
      read(ninp,100) (nn(k),k=1,n)
      write(nres,1100)
      do k=1,n
        c=c0(nn(k))
        write(nres,1001) k, nn(k), c
      enddo
  600 continue
      write(nres,*) n
      close(nres)
  100 format(i15)
 1100 format(1x,' #',2x,'k',6x,'nn(k)',11x,'C')
 1001 format(1x,i5,2x,i10,2x,e15.7)
      end
```

дало следующий результат

```
#   n= 13      C=  0.918938518
#  k      nn(k)           C
    1          10    0.9272699E+00
    2         100    0.9197388E+00
    3        1000    0.9208984E+00
    4       10000    0.9921875E+00
    5       20000    0.9062500E+00
    6       30000    0.8750000E+00
    7       40000    0.1531250E+01
    8       50000    0.1031250E+01
    9       60000    0.8750000E+00
   10       70000    0.2625000E+01
   11       80000    0.5312500E+01
   12       90000   -0.1243750E+02
   13      100000    0.1137500E+02
13
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **c1(n)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **c0(n)** и **c1(n)**.
7. Модифицировать программу для получения результатов с удвоенной точностью.

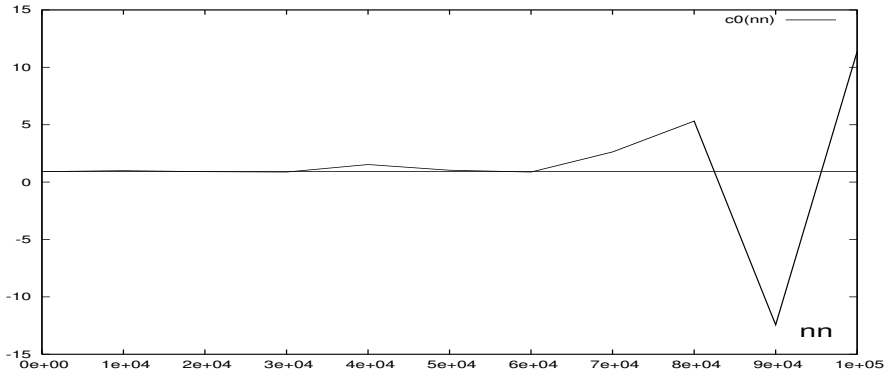

Рис. 16: График функции, табулированной c0(n).

4.15 Табулирование функции $\dfrac{2-\sqrt{4+e^{-x}}}{3-\sqrt{2(4+e^{-x})+1}}$

В некоторой задаче потребовалась табулировать указанную функцию при $\mathbf{x} = \mathbf{12}(\mathbf{0.1})\mathbf{13}$. Была составлена подпрограмма-функция $\mathbf{v0}(\mathbf{x})$:

```
      function v0(x)                          !      Файл  v0.for
      w=4+exp(-x)
      v0=(2-sqrt(w))/(3-sqrt(2*w+1))
      end
```

которая вела расчет непосредственно по формуле из заголовка. Тестирование функции проводилось программой

```
      program tsfs2p15                       !    Файл  tsfs2p15.for
      data ninp / 5 /, nres / 6 /
      pi=4*atan(1.0)
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100) x0,xn
      read(ninp,101) n
      write(nres, *) ' #   x0=',x0,'    xn=',xn,'    n=',n
      hx=(xn-x0)/n
      write(nres,1100)
      do i=0,n
        x=x0+i*hx
        r0=v0(x)
        write(nres,1001) i, x,r0
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',6x,'x',11x,'v0')
 1001 format(1x,i5,2x,2x,e15.7,e15.7)
      end
```

и на значениях аргумента из диапазона $\mathbf{t} = [-\mathbf{2}, -\mathbf{3}]$ дало на одинарной точности результаты верные в пределах семи значащих цифр мантиссы. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{12}, \mathbf{13}]$ привело к следующему:

```
#   x0=  12.    xn=  13.    n= 10
#  i      x           v0
   0      0.1200000E+02  0.6666667E+00
   1      0.1210000E+02  0.7500000E+00
   2      0.1220000E+02  0.7142857E+00
   3      0.1230000E+02  0.7142857E+00
   4      0.1240000E+02  0.6666667E+00
   5      0.1250000E+02  0.8000000E+00
   6      0.1260000E+02  0.6000000E+00
   7      0.1270000E+02  0.7500000E+00
   8      0.1280000E+02  0.7500000E+00
   9      0.1290000E+02  0.6666667E+00
  10      0.1300000E+02  0.6666667E+00
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **v1(x)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **v0(x)**, **v1(x)**.
7. Модифицировать программу для получения результатов с удвоенной точностью.

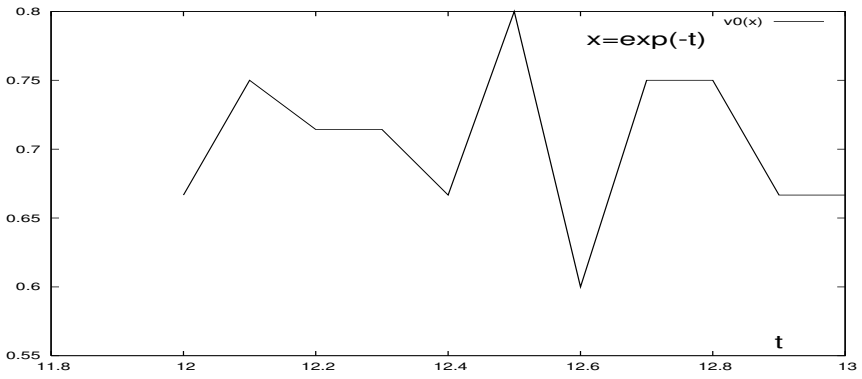

Рис. 17: График функции v0(x) при x=exp(-t) (по оси абсцисс отложено t).

4.16 Табулирование функции $\dfrac{1-\cos x}{1-\cos\dfrac{x}{2}}$

В некоторой задаче потребовалась табулировать указанную функцию для $\mathbf{x} = \mathbf{exp}(-\mathbf{t})$ при $\mathbf{t} = \mathbf{6.55}(\mathbf{0.1})\mathbf{7.55}]$. Была составлена подпрограмма-функция $\mathbf{q0}(\mathbf{x})$:

```
      function q0(x)                          !      Файл  q0.for
      q0=(1-cos(x))/(1-cos(x/2))
      end
```

которая вела расчет непосредственно по формуле из заголовка Тестирование функции проводилось программой

```
      program tsfs2p16                       !    Файл  tsfs2p16.for
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100) t0,tn
      read(ninp,101) n
      write(nres, *) ' #   t0=',t0,'    tn=',tn,'    n=',n
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
        t=t0+i*ht
        x=exp(-t)
        r0=q0(x)
        write(nres,1001) i, t,r0
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'t',12x,'q0')
 1001 format(1x,i5,2x,2x,e15.7,e15.7)
      end
```

на значениях аргумента $\mathbf{t} = [\mathbf{0.2}, \mathbf{0.3}]$ дало результаты, верные в пределах семи значащих цифр на одинарной точности. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{6.55}, \mathbf{7.55}]$ привело к следующему:

```
#   t0=  6.55000019    tn=  7.55000019    n= 10
#  i            t             q0
   0      0.6550000E+01  0.4250000E+01
   1      0.6650000E+01  0.3500000E+01
   2      0.6750000E+01  0.4000000E+01
   3      0.6850000E+01  0.4500000E+01
   4      0.6950000E+01  0.4000000E+01
   5      0.7050000E+01  0.3000000E+01
   6      0.7150000E+01  0.5000000E+01
   7      0.7250000E+01  0.4000000E+01
   8      0.7350000E+01  0.3000000E+01
   9      0.7450000E+01  0.3000000E+01
  10      0.7550000E+01  0.2000000E+01
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **q1(x)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **q0(x)**, **q1(x)**.
7. Модифицировать программу для получения результатов с удвоенной точностью и проанализировать их (в частности, выяснить при каких значениях аргумента на удвоенной точности получаются неверные результаты и продемонстрировать это).

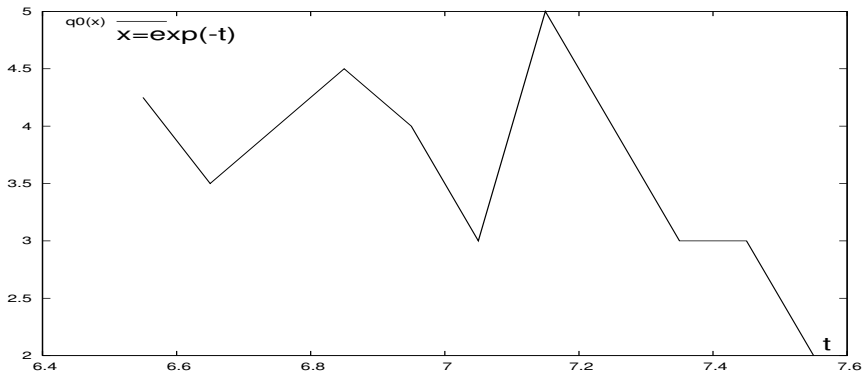

Рис. 18: График по алгоритму q0(x) при x=exp(-t) (по оси абсцисс отложено t).

4.17 Табулирование функции $\dfrac{\cos 2x - \cos x}{\sin 2x - \sin x}$

В некоторой задаче потребовалась табулировать указанную функцию для $\mathbf{x} = \mathbf{exp}(-\mathbf{t})$ при $\mathbf{t} = \mathbf{8}(\mathbf{0.1})\mathbf{9}]$. Была составлена подпрограмма-функция $\mathbf{s0}(\mathbf{x})$:

```
      function s0(x)                          !      Файл  s0.for
      s0=(cos(2*x)-cos(x))/(sin(2*x)-sin(x))
      end
```

которая вела расчет непосредственно по формуле из заголовка. Тестирование функции проводилось программой

```
      program tsfs2p17                       !    Файл  tsfs2p16.for
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn
      read(ninp,101) n
      write(nres, *) ' #   t0=',t0,'    tn=',tn,'    n=',n
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
        t=t0+i*ht
        x=exp(-t)
        r0=s0(x)
        write(nres,1001) i, t,r0
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'t',12x,'q0')
 1001 format(1x,i5,2x,2x,e15.7,e15.7)
      end
```

и на значениях аргумента из диапазона $\mathbf{t} = [\mathbf{0.5}, \mathbf{0.6}]$ дало на одинарной точности результаты верные в пределах семи значащих цифр мантиссы. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{8}, \mathbf{9}]$ привело к следующему:

```
#   t0=  8.    tn=  9.    n= 10
#  i            t            q0
   0      0.8000000E+01 -0.5330369E-03
   1      0.8100000E+01 -0.3927314E-03
   2      0.8200000E+01 -0.4340350E-03
   3      0.8300000E+01 -0.2398415E-03
   4      0.8400000E+01 -0.5301315E-03
   5      0.8500000E+01 -0.2929430E-03
   6      0.8600000E+01 -0.3237523E-03
   7      0.8700000E+01 -0.3578014E-03
   8      0.8800000E+01 -0.3954318E-03
   9      0.8900000E+01 -0.4370195E-03
  10      0.9000000E+01 -0.4829814E-03
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **s1(x)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **s0(x)**, **s1(x)**.
7. Модифицировать программу для получения результатов с удвоенной точностью и проанализировать их.

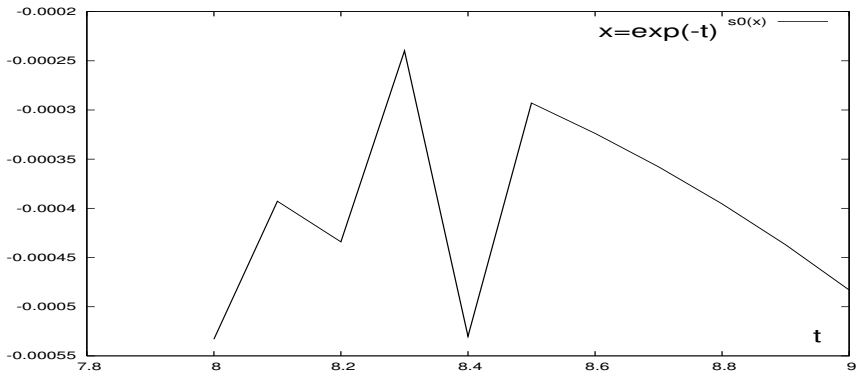

Рис. 19: График функции по алгоритму q0(x) при x=exp(-t) (по оси абсцисс отложено t).

4.18 Табулирование функции $\dfrac{1.7-(1.7^3-x)^{\frac{1}{3}}}{x}$

В некоторой задаче потребовалась табулировать указанную функцию для $\mathbf{x} = \mathbf{exp}(-\mathbf{t^2})$ при $\mathbf{t} = \mathbf{3.5}(\mathbf{0.02})\mathbf{3.8}]$. Была составлена подпрограмма-функция $\mathbf{fun0}(\mathbf{x})$:

```
      function fun0(x)                          !      Файл  fun0.for
      fun0=(1.7-(1.7**3-x)**(1.0/3))/x
      end
```

которая вела расчет непосредственно по формуле из заголовка. Тестирование функции проводилось программой

```
      program tsfs2p18                       !    Файл  tsfs2p18.for
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn
      read(ninp,101) n
      write(nres, *) ' #   t0=',t0,'    tn=',tn,'    n=',n
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
        t=t0+i*ht
        x=exp(-t*t)
        r0=fun0(x)
        write(nres,1001) i, t, r0
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'t',11x,'fun0')
 1001 format(1x,i5,2x,2x,e15.7,e15.7)
      end
```

и на значениях аргумента из диапазона $\mathbf{t} = [\mathbf{0}, \mathbf{0.6}]$ дало на одинарной точности результаты верные в пределах семи значащих цифр мантиссы. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{3.5}, \mathbf{3.8}]$ привело к следующему:

```
#   t0=  3.4000001    tn=  3.79999995    n= 10
#  i            t           fun0
   0      0.3400000E+01  0.1124597E+00
   1      0.3440000E+01  0.1149941E+00
   2      0.3480000E+01  0.1083328E+00
   3      0.3520000E+01  0.1146707E+00
   4      0.3560000E+01  0.1141578E+00
   5      0.3600000E+01  0.1013436E+00
   6      0.3640000E+01  0.1353844E+00
   7      0.3680000E+01  0.9071934E-01
   8      0.3720000E+01  0.1219694E+00
   9      0.3760000E+01  0.1645098E+00
  10      0.3800000E+01  0.2225985E+00
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **fun1(x)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **fun0(x)**, **fun1(x)**.
7. Модифицировать программу для получения результатов с удвоенной точностью и проанализировать их.

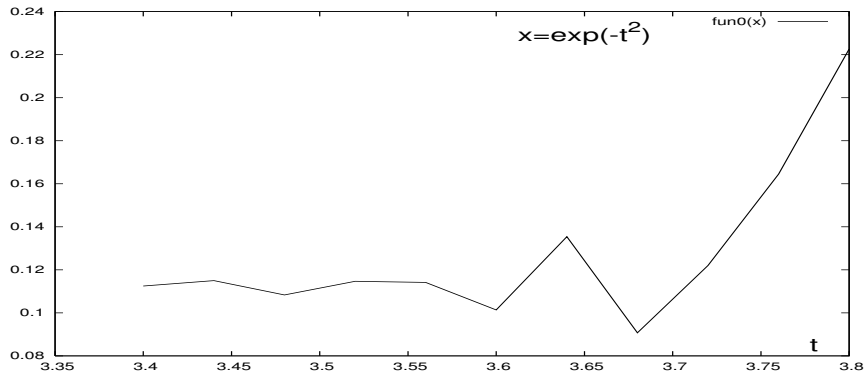

Рис. 20: График по алгоритму fun0(x) при x=exp(-t*t) (по оси абсцисс отложено t).

4.19 Табулирование функции $\dfrac{\sin x - \tan x}{4\sin^3 x/2}$

В некоторой задаче потребовалась табулировать указанную функцию для $\mathbf{x} = \mathbf{exp}(-\mathbf{t^2})$ при $\mathbf{t} = \mathbf{2.7}(\mathbf{0.1})\mathbf{2.8}]$. Была составлена подпрограмма-функция $\mathbf{sts0}(\mathbf{x})$:

```
      function sts0(x)                          !    Файл sts0.for
      sts0=(sin(x)-tan(x))/4/sin(x/2)**2
      end
```

которая вела расчет непосредственно по формуле из заголовка. Тестирование функции проводилось программой

```
      program tsfs2p19                      !    Файл  tsfs2p19.for
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn
      read(ninp,101) n
      write(nres, *) ' #   t0=',t0,'    tn=',tn,'    n=',n
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
        t=t0+i*ht
        x=exp(-t*t)
        r0=sts0(x)
        write(nres,1001) i, t, r0
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'t',12x,'sts0')
 1001 format(1x,i5,2x,2x,e15.7,e15.7)
      end
```

и на значениях аргумента из диапазона $\mathbf{t} = [\mathbf{0}, \mathbf{1.0}]$ дало на одинарной точности результаты верные в пределах семи значащих цифр мантиссы. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{2.7}, \mathbf{2.8}]$ привело к следующему:

```
#   t0=  2.70000005    tn=  2.79999995    n= 10
#  i            t             q0
   0      0.2700000E+01 -0.3750724E-03
   1      0.2710000E+01 -0.4179321E-03
   2      0.2720000E+01 -0.3105839E-03
   3      0.2730000E+01 -0.3463514E-03
   4      0.2740000E+01 -0.1931962E-03
   5      0.2750000E+01 -0.2156176E-03
   6      0.2760000E+01 -0.2407373E-03
   7      0.2770000E+01 -0.2688910E-03
   8      0.2780000E+01 -0.1502287E-03
   9      0.2790000E+01 -0.1679320E-03
  10      0.2800000E+01 -0.1877965E-03
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **sts1(x)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **sts0(x)**, **sts1(x)**.
7. Модифицировать программу для получения результатов с удвоенной точностью и проанализировать их.

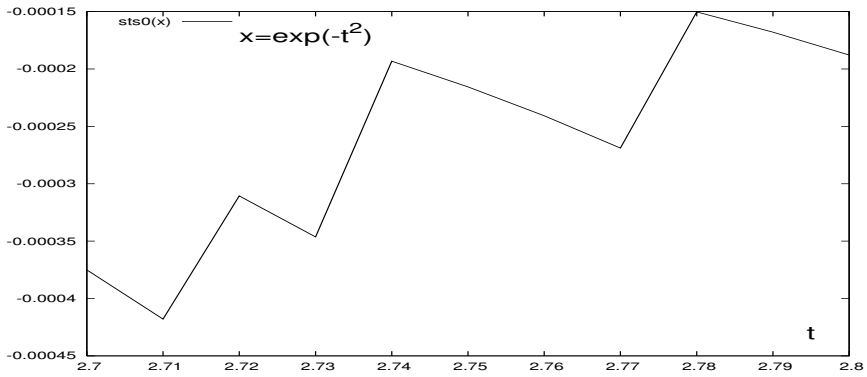

Рис. 21: График по алгоритму sts0(x) при $x = exp(-t^2)$ (по оси абсцисс отложено t).

4.20 Табулирование функции $x - \sqrt{x^2 + 5x}$.

В некоторой задаче потребовалась табулировать указанную функцию для $\mathbf{x} = \mathbf{exp}(\mathbf{t})$ при $\mathbf{t} = \mathbf{15.8}(\mathbf{0.02})\mathbf{17.8}]$. Были составлены две подпрограммы-функции $\mathbf{w0}(\mathbf{x})$ и $\mathbf{w1}(\mathbf{x})$:

```
      function w0(x)                              !    Файл w0.for
      w0=x-sqrt(x**2+5*x)
      end
      function w1(x)                              !
      w1=x*(1-sqrt(1+5/x))
      end
```

Первая вела расчет непосредственно по формуле из заголовка; вторая – по слегка модифицированной, полученной вынесением аргумента $\mathbf{x}$ из под знака корня и выделением аргумента в качестве явного сомножителя. Тестирование функций проводилось программой

```
      program tsfs2p20                        !    Файл  tsfs2p20.for
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn
      read(ninp,101) n
      write(nres, *) ' #   t0=',t0,'    tn=',tn,'    n=',n
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
        t=t0+i*ht
        x=exp(t)
        r0=w0(x)
        r1=w1(x)
        write(nres,1001) i, t, r0, r1
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'t',14x,'w0',13x,'w1')
 1001 format(1x,i5,2x,2x,e15.7,e15.7,e15.7)
      end
```

и при значениях аргумента $\mathbf{t}$ из диапазона $\mathbf{t} = [\mathbf{0}, \mathbf{1.0}]$ дало на одинарной точности результаты верные в пределах семи значащих цифр мантиссы. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{15.8}, \mathbf{17.8}]$ привело к следующему:

```
#   t0=  15.8000002    tn=  17.7999992    n= 10
#  i            t              w0             w1
   0      0.1580000E+02 -0.2500000E+01 -0.2601862E+01
   1      0.1600000E+02 -0.2000000E+01 -0.2118614E+01
   2      0.1620000E+02 -0.2000000E+01 -0.2587683E+01
   3      0.1640000E+02 -0.2000000E+01 -0.3160599E+01
   4      0.1660000E+02 -0.2000000E+01 -0.1930184E+01
   5      0.1680000E+02 -0.2000000E+01 -0.2357529E+01
   6      0.1700000E+02 -0.2000000E+01 -0.2879495E+01
   7      0.1720000E+02 -0.2000000E+01 -0.3517019E+01
   8      0.1740000E+02 -0.4000000E+01 -0.4295700E+01
   9      0.1760000E+02 -0.4000000E+01  0.0000000E+00
  10      0.1780000E+02 -0.4000000E+01  0.0000000E+00
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснитьобъективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **w2(x)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **w0(x)**, **w1(x)** и **w2(x)**.
7. Модифицировать программу для получения результатов с удвоенной точностью и проанализировать их.

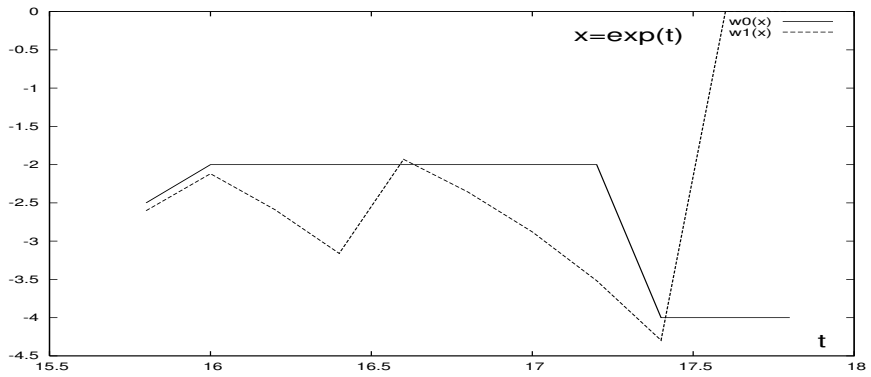

Рис. 22: Графики функции, табулированной алгоритмами w0(x) и w1(x) при x=exp(t) (по оси абсцисс отложено t).

4.21 Табулирование функции $\sqrt{tg^2x + \frac{1.23}{cos(x)}} - tg(x)$.

В некоторой задаче потребовалась табулировать указанную функцию для $x = \frac{\pi}{2} - e^{-t}$ при $t = 12.3(0.4)16.3]$. Была составлена подпрограмма-функции **w0**(**x**):

```
      function tg0(x)                              !    Файл tg0.for
      tg=tan(x)
      tg2=tg*tg
      tg0=sqrt(tg2+1.23/cos(x))-tg
      return
      end
```

которая вела расчет непосредственно по формуле из заголовка. Тестирование функций проводилось программой

```
      program tsfs2p21                        !    Файл  tsfs2p21.for
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn
      read(ninp,101) n
      write(nres, *) ' #   t0=',t0,'    tn=',tn,'    n=',n
      pi2=2*atan(1.0)
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
        t=t0+i*ht
        x=pi2-exp(-t)
        r0=tg0(x)
        write(nres,1001) i, t, r0
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'t',14x,'tg0')
 1001 format(1x,i5,2x,2x,e15.7,e15.7)
      end
```

и при значениях аргумента **t** из диапазона $\mathbf{t} = [\mathbf{0}, \mathbf{1.0}]$ дало на одинарной точности результаты верные в пределах семи значащих цифр мантиссы. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{12.3}, \mathbf{16.3}]$ привело к следующему:

```
#   t0=  12.3000002    tn=  16.2999992    n= 10
#  i            t              tg0
    0      0.1230000E+02  0.6093750E+00
    1      0.1270000E+02  0.6250000E+00
    2      0.1310000E+02  0.6250000E+00
    3      0.1350000E+02  0.6250000E+00
    4      0.1390000E+02  0.6250000E+00
    5      0.1430000E+02  0.6250000E+00
    6      0.1470000E+02  0.7500000E+00
    7      0.1510000E+02  0.5000000E+00
    8      0.1550000E+02  0.5000000E+00
    9      0.1590000E+02  0.1000000E+01
   10      0.1630000E+02  0.1000000E+01
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **tg1(x)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **tg0(x)** и **tg1(x)**.
7. Модифицировать программу для получения результатов с удвоенной точностью и проанализировать их.

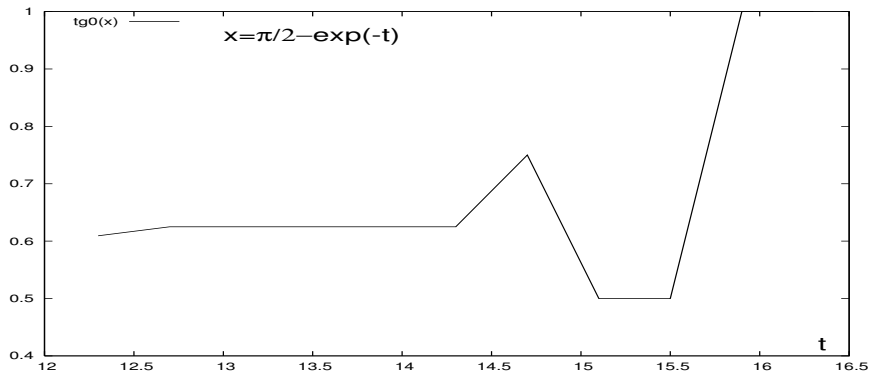

Рис. 23: График функции, табулированной алгоритмом tg0(x) при $x = \frac{\pi}{2} - \exp -t$ (по оси абсцисс отложено t).

4.22 Табулирование функции $\frac{cos(2x)}{sin(x)-cos(x)}$.

В некоторой задаче потребовалась табулировать указанную функцию для $x = \frac{\pi}{4} - e^{-t}$ при $t = 15(0.2)17]$. Была составлена подпрограмма-функции **w0**(**x**):

```
      function w0(x)                                !    Файл w0.for
      w0=cos(2*x)/(sin(x)-cos(x))
      end
```

которая вела расчет непосредственно по формуле из заголовка. Тестирование функций проводилось программой

```
      program tsfs2p22                         !    Файл  tsfs2p22.for
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn
      read(ninp,101) n
      write(nres, *) ' #   t0=',t0,'    tn=',tn,'    n=',n
      pi4=atan(1.0)
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
        t=t0+i*ht
        x=pi4-exp(-t)
        r0=w0(x)
        write(nres,1001) i, t, r0
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'t',14x,'w0')
 1001 format(1x,i5,2x,2x,e15.7,e15.7)
      end
```

и при значениях аргумента **t** из диапазона $\mathbf{t} = [\mathbf{0}, \mathbf{1.0}]$ дало на одинарной точности результаты верные в пределах семи значащих цифр мантиссы. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{15}, \mathbf{17}]$ привело к следующему:

```
#   t0=  15.    tn=  17.    n= 10
#  i            t              w0
   0      0.1500000E+02 -0.1544441E+01
   1      0.1520000E+02 -0.1453329E+01
   2      0.1540000E+02 -0.1316661E+01
   3      0.1560000E+02 -0.1316661E+01
   4      0.1580000E+02 -0.1633322E+01
   5      0.1600000E+02 -0.1633322E+01
   6      0.1620000E+02 -0.1633322E+01
   7      0.1640000E+02 -0.1266645E+01
   8      0.1660000E+02 -0.1266645E+01
   9      0.1680000E+02 -0.1266645E+01
  10      0.1700000E+02 -0.1266645E+01
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **w1(x)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **w0(x)** и **w1(x)**.
7. Модифицировать программу для получения результатов с удвоенной точностью и проанализировать их.

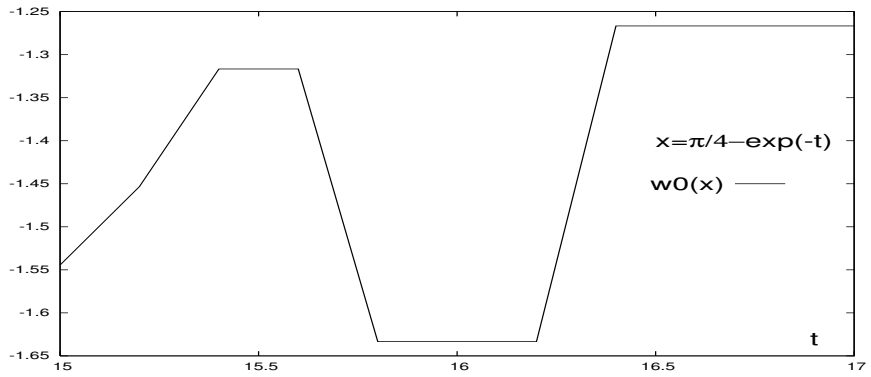

Рис. 24: Графики функции, табулированной алгоритмом w0(x) при $x = \frac{\pi}{4} - \exp -t$ (по оси абсцисс отложено t).

4.23 Табулирование функции $\dfrac{\sin x}{\sqrt{2(1-\cos x)}}$.

В некоторой задаче потребовалась табулировать указанную функцию для $x = e^{-t}$ при $t = 7.3(0.1)8.3]$. Была составлена подпрограмма-функции **w0(x)**:

```
      function w0(x)                              !    Файл w0.for
      w0=sin(x)/sqrt(2*(1-cos(x)))
      end
```

которая вела расчет непосредственно по формуле из заголовка. Тестирование функций проводилось программой

```
      program tsfs2p23                        !    Файл  tsfs2p23.for
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn
      read(ninp,101) n
      write(nres, *) ' #   t0=',t0,'    tn=',tn,'    n=',n
      pi4=atan(1.0)
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
        t=t0+i*ht
        x=exp(-t)
        r0=w0(x)
        write(nres,1001) i, t, r0
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'t',14x,'w0')
 1001 format(1x,i5,2x,2x,e15.7,e15.7,e15.7)
      end
```

и при значениях аргумента **t** из диапазона $\mathbf{t} = [\mathbf{0.5}, \mathbf{0.6}]$ дало на одинарной точности результаты верные в пределах семи значащих цифр мантиссы. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{7.3}, \mathbf{8.3}]$ привело к следующему:

```
#   t0=  7.30000019    tn=  8.30000019    n= 10
#  i            t              w0
   0      0.7300000E+01  0.9782844E+00
   1      0.7400000E+01  0.1022127E+01
   2      0.7500000E+01  0.9248593E+00
   3      0.7600000E+01  0.1024924E+01
   4      0.7700000E+01  0.9273897E+00
   5      0.7800000E+01  0.1186719E+01
   6      0.7900000E+01  0.1073788E+01
   7      0.8000000E+01  0.9716036E+00
   8      0.8100000E+01  0.8791429E+00
   9      0.8200000E+01  0.7954819E+00
  10      0.8300000E+01  0.7197815E+00
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **w1(x)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **w0(x)** и **w1(x)**.
7. Модифицировать программу для получения результатов с удвоенной точностью и проанализировать их.

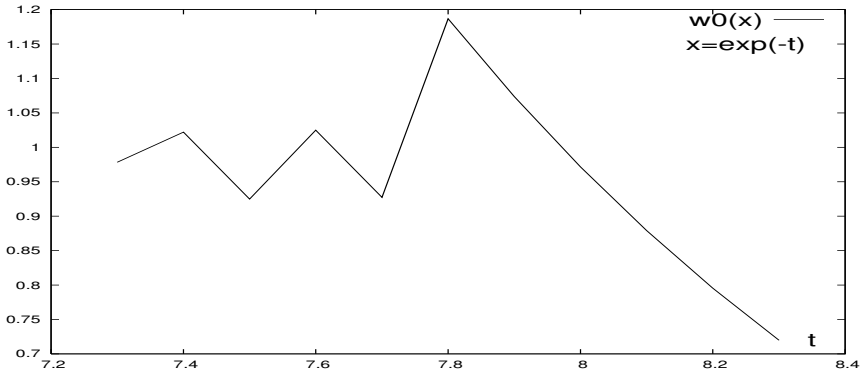

Рис. 25: Графики функции, табулированной алгоритмом w0(x) при $x = e^{-t}$.

4.24 Табулирование функции $\sqrt{\frac{1-\cos x}{1+\cos x)}}$.

В некоторой задаче потребовалась табулировать указанную функцию для $x = e^{-t}$ при $t = 7.3(0.1)8.3]$. Была составлена подпрограмма-функции **w0(x)**:

```
      function w0(x)                              !    Файл w0.for
      w0=sqrt( (1-cos(x)) / (1+cos(x)) )
      end
```

которая вела расчет непосредственно по формуле из заголовка. Тестирование функций проводилось программой

```
      program tsfs2p24                        !    Файл  tsfs2p24.for
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn
      read(ninp,101) n
      write(nres, *) ' #   t0=',t0,'    tn=',tn,'    n=',n
      pi4=atan(1.0)
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
        t=(t0+i*ht)
        x=exp(-t)
        r0=w0(x)
        write(nres,1001) i, t, r0
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'t',14x,'w0')
 1001 format(1x,i5,2x,2x,e15.7,e15.7)
      end
```

и при значениях аргумента **t** из диапазона $\mathbf{t} = [\mathbf{0.5}, \mathbf{0.6}]$ дало на одинарной точности результаты верные в пределах семи значащих цифр мантиссы. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{7.3}, \mathbf{8.3}]$ привело к следующему:

```
#   t0=  7.30000019    tn=  8.30000019    n= 10
#  i            t              w0
    0      0.7300000E+01  0.3452670E-03
    1      0.7400000E+01  0.2990100E-03
    2      0.7500000E+01  0.2990100E-03
    3      0.7600000E+01  0.2441406E-03
    4      0.7700000E+01  0.2441406E-03
    5      0.7800000E+01  0.1726335E-03
    6      0.7900000E+01  0.1726335E-03
    7      0.8000000E+01  0.1726335E-03
    8      0.8100000E+01  0.1726335E-03
    9      0.8200000E+01  0.1726335E-03
   10      0.8300000E+01  0.1726335E-03
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **w1(x)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **w0(x)** и **w1(x)**.
7. Модифицировать программу для получения результатов с удвоенной точностью и проанализировать их.

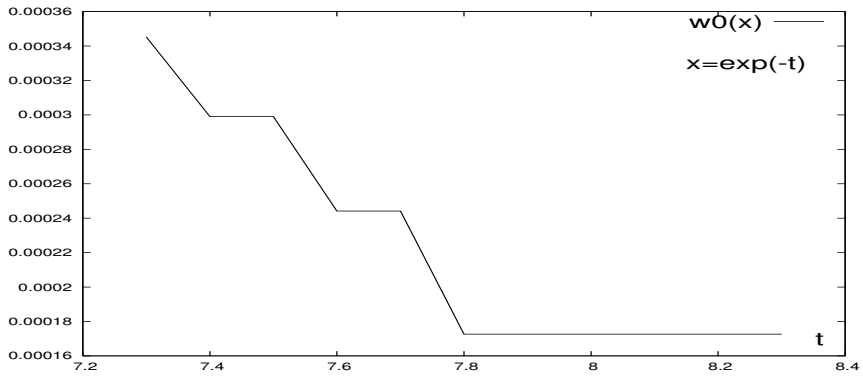

Рис. 26: Графики функции, табулированной алгоритмом w0(x) при $x = e^{-t}$.

4.25 Табулирование функции $\dfrac{\sin 3x - 3\sin x}{4\sin^3 x}$.

В некоторой задаче потребовалась табулировать указанную функцию для $x = e^{-t}$ при $t = 9.3(0.1)10.3]$. Была составлена подпрограмма-функции **w0**(**x**):

```
      function w0(x)                              !    Файл w0.for
      w0=(sin(3*x)-3*sin(x))/4/sin(x)**3
      end
```

которая вела расчет непосредственно по формуле из заголовка. Тестирование функций проводилось программой

```
      program tsfs2p25                        !    Файл  tsfs2p25.for
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn
      read(ninp,101) n
      write(nres, *) ' #   t0=',t0,'    tn=',tn,'    n=',n
      pi4=atan(1.0)
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
        t=(t0+i*ht)
        x=exp(-t)
        r0=w0(x)
        write(nres,1001) i, t, r0
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'t',14x,'w0')
 1001 format(1x,i5,2x,2x,e15.7,e15.7)
      end
```

и при значениях аргумента **t** из диапазона $\mathbf{t} = [\mathbf{0.9}, \mathbf{1.0}]$ дало на одинарной точности результаты верные в пределах семи значащих цифр мантиссы. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{9.3}, \mathbf{10.3}]$ привело к следующему:

```
#   t0=   9.300000         tn=   10.30000         n=          10
#  i            t              w0
    0      0.9300000E+01  0.0000000E+00
    1      0.9400001E+01 -0.6426364E+01
    2      0.9500000E+01 -0.4337335E+01
    3      0.9600000E+01 -0.5854796E+01
    4      0.9700000E+01 -0.7903134E+01
    5      0.9800000E+01  0.5334063E+01
    6      0.9900001E+01  0.0000000E+00
    7      0.1000000E+02 -0.1943858E+02
    8      0.1010000E+02 -0.2623937E+02
    9      0.1020000E+02 -0.1770970E+02
   10      0.1030000E+02 -0.2390561E+02
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **w1(x)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **w0(x)** и **w1(x)**.
7. Модифицировать программу для получения результатов с удвоенной точностью и проанализировать их.

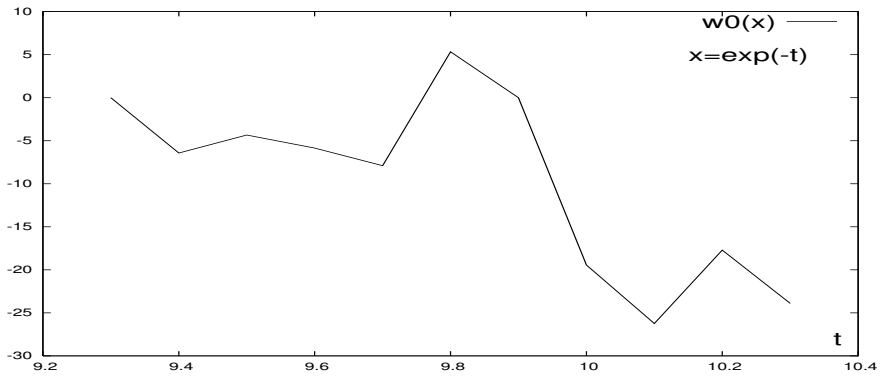

Рис. 27: Графики функции, табулированной алгоритмом w0(x) при $x = e^{-t}$.

4.26 Табулирование функции $\dfrac{\cos 4x - 1}{8\cos^4 x - 8\cos^2 x}$.

В некоторой задаче потребовалась табулировать указанную функцию для $x = e^{-t}$ при $t = 9.3(0.1)10.3]$. Была составлена подпрограмма-функции **w0(x)**:

```
      function w0(x)                                !    Файл w0.for
      w0=(cos(4*x)-1)/(8*cos(x)**4-8*cos(x)**2)
      end
```

которая вела расчет непосредственно по формуле из заголовка. Тестирование функций проводилось программой

```
      program tsfs2p26                        !    Файл  tsfs2p26.for
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn
      read(ninp,101) n
      write(nres, *) ' #   t0=',t0,'    tn=',tn,'    n=',n
      pi4=atan(1.0)
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
        t=(t0+i*ht)
        x=exp(-t)
        r0=w0(x)
        write(nres,1001) i, t, r0
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'t',14x,'w0',14x)
 1001 format(1x,i5,2x,2x,e15.7,e15.7)
      end
```

и при значениях аргумента **t** из диапазона $\mathbf{t} = [\mathbf{0.9}, \mathbf{1.0}]$ дало на одинарной точности результаты верные в пределах семи значащих цифр мантиссы. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{9.3}, \mathbf{10.3}]$ привело к следующему:

```
#   t0=   9.300000         tn=   10.30000         n=          10
#  i            t              w0
    0      0.9300000E+01  0.0000000E+00
    1      0.9400001E+01 -0.6426364E+01
    2      0.9500000E+01 -0.4337335E+01
    3      0.9600000E+01 -0.5854796E+01
    4      0.9700000E+01 -0.7903134E+01
    5      0.9800000E+01  0.5334063E+01
    6      0.9900001E+01  0.0000000E+00
    7      0.1000000E+02 -0.1943858E+02
    8      0.1010000E+02 -0.2623937E+02
    9      0.1020000E+02 -0.1770970E+02
   10      0.1030000E+02 -0.2390561E+02
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **w1(x)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **w0(x)** и **w1(x)**.
7. Модифицировать программу для получения результатов с удвоенной точностью и проанализировать их.

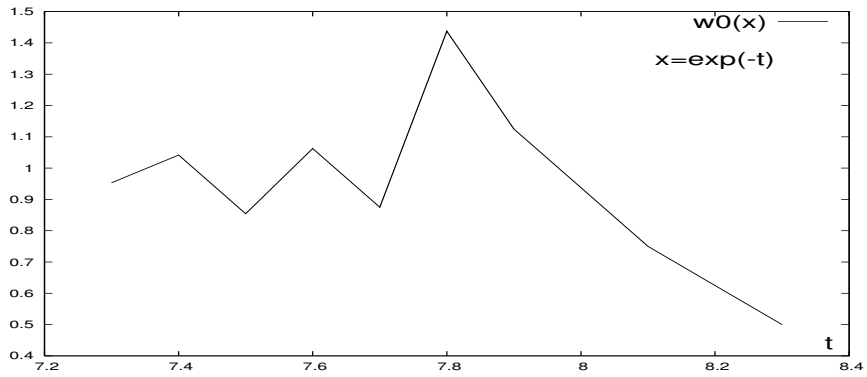

Рис. 28: Графики функции, табулированной алгоритмом w0(x) при $x = e^{-t}$.

4.27 Табулирование функции $\dfrac{\sin^2(0.5+x)-\sin^2 0.5}{\cos^2(0.5+x)-\cos^2 0.5}$.

В некоторой задаче потребовалась табулировать указанную функцию для $x = e^{-t}$ при $t = 15.2(0.1)16.2]$. Была составлена подпрограмма-функции **w0**(**x**):

```
      function w0(x)                                !    Файл w0.for
      w0=(sin(0.5+x)**2-sin(0.5)**2)/(cos(0.5+x)**2-cos(0.5)**2)
      end
```

которая вела расчет непосредственно по формуле из заголовка. Тестирование функций проводилось программой

```
      program tsfs2p27                          !    Файл  tsfs2p27.for
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn
      read(ninp,101) n
      write(nres, *) ' #   t0=',t0,'    tn=',tn,'    n=',n
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
        t=(t0+i*ht)
        x=exp(-t)
        r0=w0(x)
        write(nres,1001) i, t, r0
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'t',14x,'w0')
 1001 format(1x,i5,2x,2x,e15.7,e15.7)
      end
```

и при значениях аргумента **t** из диапазона $\mathbf{t} = [\mathbf{1.52}, \mathbf{1.62}]$ дало на одинарной точности результаты верные в пределах семи значащих цифр мантиссы. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{15.2}, \mathbf{16.2}]$ привело к следующему:

```
#   t0=  15.1999998    tn=  16.2000008    n= 10
#  i            t              w0
   0      0.1520000E+02 -0.8520044E+00
   1      0.1530000E+02 -0.7236916E+00
   2      0.1540000E+02 -0.1486786E+01
   3      0.1550000E+02 -0.1223174E+01
   4      0.1560000E+02 -0.1223174E+01
   5      0.1570000E+02 -0.9595627E+00
   6      0.1580000E+02 -0.9595627E+00
   7      0.1590000E+02 -0.6959509E+00
   8      0.1600000E+02 -0.6959509E+00
   9      0.1610000E+02 -0.6959509E+00
  10      0.1620000E+02 -0.5641450E+00
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **w1(x)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **w0(x)** и **w1(x)**.
7. Модифицировать программу для получения результатов с удвоенной точностью и проанализировать их.

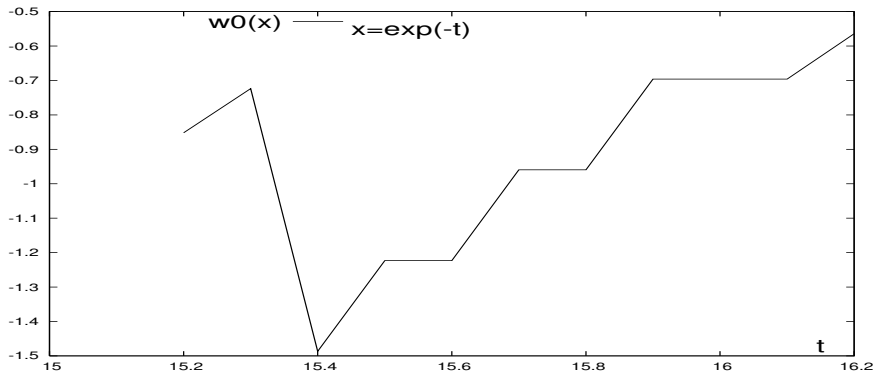

Рис. 29: Графики функции, табулированной алгоритмом w0(x) при $x = e^{-t}$.

4.28 Расчет постоянной Эйлера $\gamma = 0.57721566490$

Некто с целью проверить значение постоянной Эйлера, приводимое в справочниках (см., например, [14][стр.14]), составил подпрограмму-функции **cs(x)**:

```
      function cs(n)                                  !    Файл cs.for
      s=0.0;           ! Функция cs(n) вычисляет приближение к постоянной
      do k=1,n         ! Эйлера  через  разность  между  n-ой  частичной
        s=s+1.0/k      ! суммой гармонического ряда и ln(n).
      enddo
      cs=s-alog(float(n))
      end
```

которая вела расчет по формуле (см., например, [14][6.1.3])

$$\gamma(\mathbf{n}) = \sum_{\mathbf{k=1}}^{\mathbf{n}} \frac{\mathbf{1}}{\mathbf{k}} - \ln \mathbf{n}$$

Пользователь полагал, что увеличивая **n** число слагаемых в ней, сможет получить сколько угодно верных значащих цифр постоянной. Тестирование функций проводилось программой

```
      program tsfs2p28                       !    Файл  tsfs2p28.for
      data ninp / 5 /, nres / 6 /
      dimension nn(20)
      data nn /    100,     1000,      5000,   10000,   20000,
     >      30000,   40000,    50000,     60000,   70000,   80000,
     >      90000,  100000,   200000,    500000,  800000, 1000000,
     >    2000000, 5000000, 10000000/
      data gamma /0.5772 15664 90153 28606 06512/
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,101)  n0, n1
      write(nres, *) ' #   n0=',n0,'    n1=',n1
      write(nres,1100)
      do i=n0,n1
        n=nn(i)
        r0=cs(n)
        write(nres,1001) i, n, r0
      enddo
      close(nres)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'n',14x,'r0')
 1001 format(1x,i5,2x,2x,i15,e15.7)
      end
```

которая вводила из файла номера элементов массива, хранящего набор испытываемых значений **n**. Результат оказался следующим:

```
#   n0=           2     n1=          20
#  i            n              r0
   2             1000  0.5777230E+00
   3             5000  0.5773211E+00
   4            10000  0.5772724E+00
   5            20000  0.5772696E+00
   6            30000  0.5772114E+00
   7            40000  0.5772114E+00
   8            50000  0.5771437E+00
   9            60000  0.5774698E+00
  10            70000  0.5770512E+00
  11            80000  0.5770226E+00
  12            90000  0.5773859E+00
  13           100000  0.5779257E+00
  14           200000  0.5766840E+00
  15           500000  0.5683289E+00
  16           800000  0.5742559E+00
  17          1000000  0.5418472E+00
  18          2000000  0.8023748E+00
  19          5000000 -0.2126598E-01
  20         10000000 -0.7144127E+00
```

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **cs1(n)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **cs(n)**, **cs1(n)**.
7. Модифицировать программу для получения результатов с удвоенной

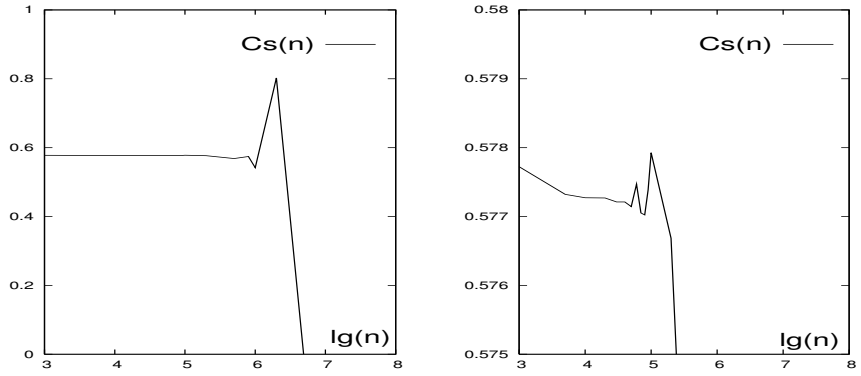

Рис. 30: Графики функции, табулированной алгоритмом cs(n).

4.29 Проверка формулы $\dfrac{a^3-b^3}{a-b}=a^2+a\cdot b+b^2$

Школьник проверяет численно на компьютере правильность алгебраической формулы из заголовка. Расчет ее обеих частей оформлен функциями **ab0(a,b)** и ab1(a,b):

```
c                                                    файл ab.for
      function ab0(a,b)
      a3=a*a*a
      b3=b*b*b
      d=a-b
      ab0 =(a3-b3)/d
      end
      function ab1(a,b)
      ab1 =a*a+a*b+b*b
      end
```

Программа для значения **b** вычисляла **a** по формуле $\mathbf{a}=\mathbf{b}+\mathbf{x}$, а в качестве приращения **x** брались точки равномерного дробления промежутка $[\mathbf{x0},\mathbf{x1}]$. Количество шагов дробления (**n**), концевые точки промежутка (**x0** и **x1**) и значение **b**,вводятся из файла.

```
      program tsfs2p11a
      data ninp / 5 /, data nres / 6 /
      open(unit=ninp,file='input')
      open(unit=nres,file='result')
      read(ninp,100) n
      read(ninp,101) x0, x1, b
      write(nres,1000) b
      h=(x1-x0)/n
      do i=1,n
        x=x0+(i-1)*h;  a=b+x
        r0=ab0(a,b)
        r1=ab1(a,b)
        write(nres,1101) x, r0, r1
      enddo
      close(nres)
  100 format(i10)
  101 format(e10.3)
 1000 format(1x,'# b=',e15.7,5x,' a=b+x'/1x,'#',50('-')/
     > 1x,'#',7x,'x',12x,'(a3-b3)/(a-b)',4x,'a^2+ab+b^2)/x'/
     > 1x,'#',50('-'))
 1101 format(1x,e17.7,e17.7,e17.7)
      end
```

Обнаружилось, что при $\mathbf{b}=\mathbf{1.7}$ и $\mathbf{x}\in[\mathbf{10^{-7}},\mathbf{10^{-6}}$ результаты, полученные по формулам левой и правой частей, которые аналитически эквивалентны друг другу, далеко не одинаковы. В то же время при $\mathbf{x}\in[\mathbf{2},\mathbf{3}]$ результаты расчета правой левой частей совпадали: Помогите школьнику понять получаемые результаты:

```
# b=  0.1900000E+01      a=b+x
#-------------------------------------------------
#       x            (a3-b3)/(a-b)    a^2+ab+b^2)/x
#-------------------------------------------------
    0.1000000E-06    0.1200000E+02    0.1083000E+02
    0.2000000E-06    0.1000000E+02    0.1083000E+02
    0.3000000E-06    0.1066667E+02    0.1083000E+02
    0.4000000E-06    0.1066667E+02    0.1083000E+02
    0.5000000E-06    0.1100000E+02    0.1083000E+02
    0.6000000E-06    0.1120000E+02    0.1083000E+02
    0.7000000E-06    0.1066667E+02    0.1083000E+02
    0.8000000E-06    0.1085714E+02    0.1083000E+02
    0.9000000E-06    0.1100000E+02    0.1083000E+02
    0.1000000E-05    0.1100000E+02    0.1083000E+02
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Почему для расчета выгоднее алгоритм функции **ab1**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод на один рисунок двух графиков, соответствующих таблице.
7. Модифицировать программу для получения результатов с удвоенной точностью.

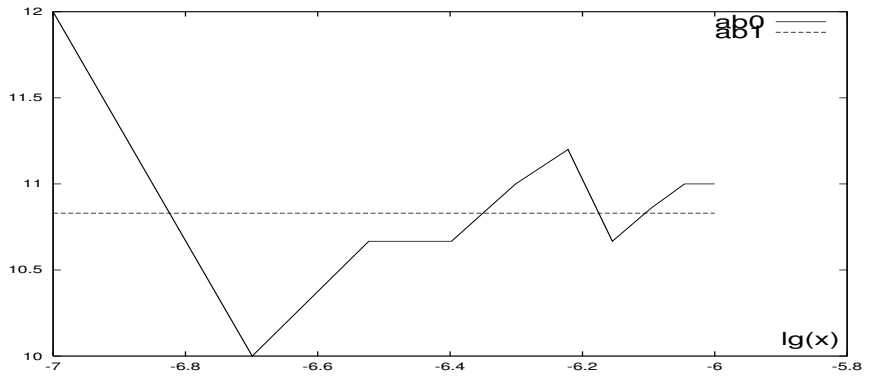

Рис. 31: График функции, табулированной на основе вызовов ab0 и ab1.

4.30 Табулирование функции $\dfrac{\cos x - 1 + \frac{x^2}{2} - \frac{x^4}{24} + \frac{x^6}{720}}{\frac{\sin x}{x} - 1 + \frac{x^2}{6} - \frac{x^4}{120}}$.

В некоторой задаче потребовалась табулировать указанную функцию для $x = e^{-t}$ при $t = 15.2(0.1)16.2]$. Была составлена подпрограмма-функции **w0**(**x**):

```
      function w0(x)                                    !    Файл w0.for
      x2=x*x
      a=cos(x) - 1 + x2* 0.5*(1 - x2/12 * ( 1 - x2 / 30))
      b=sin(x)/x - 1+x2/6 * (1-x2/20*(1-x2/42))
      w0=a/b
      end
```

которая вела расчет непосредственно по формуле из заголовка. Тестирование функций проводилось программой

```
      program tsfs2p30                        !    Файл  tsfs2p30.for
      data ninp / 5 /, nres / 6 /
      open(unit=ninp, file='input')
      open(unit=nres, file='result')
      read(ninp,100)  t0, tn
      read(ninp,101) n
      write(nres, *) ' #   t0=',t0,'    tn=',tn,'    n=',n
      pi4=atan(1.0)
      ht=(tn-t0)/n
      write(nres,1100)
      do i=0,n
        t=(t0+i*ht)
        x=exp(-t)
        r0=w0(x)
        write(nres,1001) i, t, r0
      enddo
      close(nres)
  100 format(e15.7)
  101 format(i15)
 1100 format(1x,' #',2x,'i',12x,'t',14x,'w0')
 1001 format(1x,i5,2x,2x,e15.7,e15.7)
      end
```

и при значениях аргумента **t** из диапазона $\mathbf{t} = [-\mathbf{1.0}, -\mathbf{2.0}]$ (то есть при $\mathbf{x} = \mathbf{e}$ и $\mathbf{x} = \mathbf{e^2}$) дало на одинарной точности результаты верные в пределах семи значащих цифр мантиссы. Однако, тестирование функции на требуемом рабочем диапазоне $\mathbf{t} \in [\mathbf{15.2}, \mathbf{16.2}]$ привело к следующему:

```
#   t0=   1.000000         tn=   2.000000         n=          10
#  i            t              w0
    0      0.1000000E+01  0.5782932E+01
    1      0.1100000E+01  0.6694451E+01
    2      0.1200000E+01  0.6426376E+01
    3      0.1300000E+01  0.7200233E+01
    4      0.1400000E+01  0.3847253E+01
    5      0.1500000E+01  0.1791706E+01
    6      0.1600000E+01  0.3489712E+01
    7      0.1700000E+01  0.3166201E+01
    8      0.1800000E+01  0.3158754E+01
    9      0.1900000E+01  0.2960747E+01
   10      0.2000000E+01  0.3008335E+01
```

Насколько они верны (или неверны) и почему?

1. Письменно сформулировать свое отношение к полученному результату.
2. Объяснить объективную и субъективную причины его появления.
3. Преобразовать расчетную формулу так, чтобы результат был верен.
4. Написать соответствующую новой схеме расчета функцию **w1(x)**.
5. Обеспечить наглядный вывод результатов расчета по обеим формулам в одну таблицу.
6. Включить в **make**-файл вывод рисунка с графиками **w0(x)** и **w1(x)**.
7. Модифицировать программу для получения результатов с удвоенной точностью и проанализировать их.

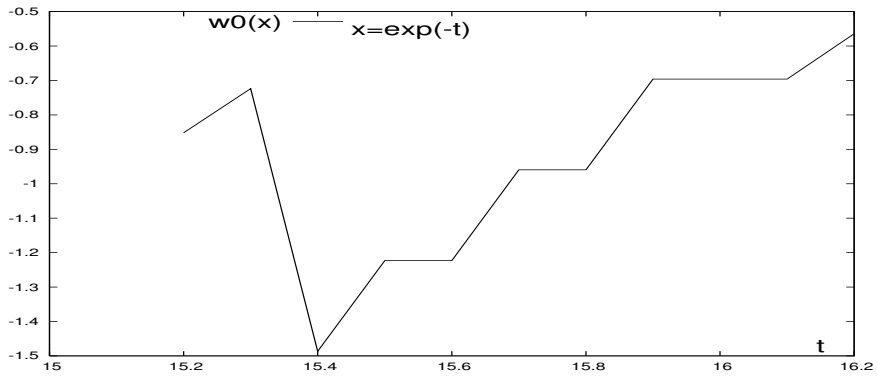

Рис. 32: Графики функции, табулированной алгоритмом w0(x) при $x = e^{-t}$ (по оси абсцисс отложено t).

Список литературы

[1] Бартеньев О.В. 1999. Фортран для студентов. – М.:"ДИАЛОГ – МИФИ", – 400 с.

[2] Бартеньев О.В. 2000. Современный ФОРТРАН. "3-е изд., доп. и перераб. – М.; ДИАЛОГ – МИФИ, – 448 с.

[3] Рыжиков Ю.И. 2004. Современный ФОРТРАН: Учебник. – СПб.: КОРОНА принт, –288 с.

[4] Немнюгин М.А., Стесик О.Л. 2004. Современный ФОРТРАН: Самоучитель. – СПб.: БХВ-Петербург, 496 с.

[5] Романовская Л.М., Русс Т.В., Свитковский С.Г. Программирование в среде Си для ПЭВМ ЕС. – М.: Финансы и статистика, 1991, – 352 с.:ил.

[6] Касаткин А.И., Вальвачев А.Н. 1992. Профессиональное программирование на языке СИ: От Turbo C к Borland C++: Справ. пособие; Под общ. ред. А.И. Касаткина. – Мн.: Выш.шк., – 240 с.

[7] Шилдт Г. 2002. - Самоучитель C++: Пер. с англ. – 3-е изд. – СПб.: БХВ–Петербург, – 688 с.

[8] Гриффитс А. 2004. gcc. Настольная книга пользователей, программистов и системных администраторов. Пер. с англ./Артур Гриффитс. – К.: “ТИД“ДС”, –624 с.

[9] Клюшин Д.А. 2004. Полный курс C++. Професстональная работа. – М.: Издательский дом “Вильямс” – 624 с. : ил.

[10] Игнатов В. 2000. Эффективное использование GNU Make.

[11] Richard M.Stallman, Roand McGrath GNU Make Программа управления компиляцией. GNU make Версия 3.79 Апрель 2000. перевод (C) Владимира Игнатова *http* : *//linux.yaroslavl.ru/docs/prog/gnu_make_3 – 79_russian_manual.html*

[12] Левин М. 2005. СИ++: Самоучитель / Максим Левин. – М.: ЗАО “Новый издательский дом”, – 176с.

[13] Кристиан. 1985. Введение в ОС UNIX – М. ???? с.

[14] Справочник по специальным функциям с формулами, графиками и таблицами. Под редакцией М. Абрамовица и И. Стиган. 1979. Москва “Наука” Главная редакция физико-математической литературы. Перевод с английского под редакцией В.А. Диткина и Л.Н. Кармазиной. М., 832 стр. с илл.