САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА АСТРОФИЗИКИ

А.Б. Шнейвайс

ПРАКТИКА ПРОГРАММИРОВАНИЯ

(приближённое вычисление интегралов)

ФОРТРАН

Второй семестр

для студентов специалитета «АСТРОНОМИЯ»

САНКТ-ПЕТЕРБУРГ
2021

Рецензенты: кандидат техн. наук, доцент В.Б. Синильщиков
(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова)
кандидат физ.-мат. наук, доцент В.Б. Титов
(СПбГУ)

Печатается по постановлению
Учебно-методической комиссии по укрупнённой группе
направлений и специальностей 03.00.00"Физика и астрономия"

Шнейвайс А.Б.

Практика программирования (приближённое вычисление интегралов) ФОРТРАН Второй семестр для студентов специалитета «АСТРОНОМИЯ»: Учебное пособие –СПб,2021.–63 с.

Учебное пособие «ПРАКТИКА ПРОГРАММИРОВАНИЯ (ВЫЧИСЛЕНИИЕ ИНТЕГРАЛОВ)» предназначено для студентов по специальности «АСТРОНОМИЯ». Рассматриваются алгортимы формул средних прямоугольников, трапеций, Симпсона, Гаусса, метода двойного пересчёта. Приводятся некоторые исходные тексты известных процедур расчёта узлов и весов квадратур гауссова типа и даются примеры их использования для расчёта интегралов. Приводятся результаты применения к расчёту интегралов адаптивной процедуры QUANC.

Содержание

1 Введение

Процедуры численного расчёта интегралов широко востребованы в в самых разных областях математики, механики, физики, астрономии.

Алгоритмы процедур численного интегрирования полезны и при освоении языков программирования, нацеленных на решение задач вычислительного характера, так как предоставляют обучающимся возможность закрепить навыки по использованию операторов цикла, массивов, процедур и модулей.

Конечно, при решении различных прикладных задач нередко пользуются многочисленными пакетами программного обеспечения, которые содержат немало соответствующих процедур. Однако, как правило, эти пакеты предполагают наличие у пользователя определённого опыта, который можно приобрести, программируя известные простые алгоритмы вычисления интегралов и тестируя их. Обычно перед использованием в предполагаемом проекте процедуры из какого-нибудь известного пакета её, тем не менее, полезно протестировать отдельно. И тут может помочь опыт тестирования простых процедур.

Исходные тексты программ, приведённых в данном пособии, написаны на языке программирования ФОРТРАН. Современный ФОРТРАН [1, 2, 3, 4, 5] предоставляет пользователю средства программирования, которых не было у первоначальных версий ФОРТРАНа. Однако, студенты могут привлекаться к дополнению или модификации пакетов программ, написанных программистами старшего поколения. Поэтому некоторые из исходных тестов даны в стиле ФОРТРАНа-77).

Представляется, что простые примеры, излагаемые далее, позволят первокурсникам с одной стороны освоить необходимые приёмы программирования, а с другой приобрести некоторый полезный опыт численного расчёта интегралов.

1.1 Постановка задачи.

$$\mathbf{S} \equiv \int_{\mathbf{a}}^{\mathbf{b}} \mathbf{f}(\mathbf{x})\mathbf{dx}$$

Доказано, что для любой непрерывной и однозначной на промежутке $[\mathbf{a}, \mathbf{b}]$ функции $\mathbf{f}(\mathbf{x})$ существует **первообразная функция** $\mathbf{F}(\mathbf{x})$ такая, что

$$\mathbf{F}'(\mathbf{x}) = \mathbf{f}(\mathbf{x}) \quad \text{или что то же} \quad \frac{\mathbf{dF}(\mathbf{x})}{\mathbf{dx}} = \mathbf{f}(\mathbf{x}) \quad \text{или} \quad \mathbf{dF}(\mathbf{x}) = \mathbf{f}(\mathbf{x})\mathbf{dx}.$$

Так что

$$\mathbf{S} = \int_{\mathbf{a}}^{\mathbf{b}} \mathbf{f}(\mathbf{x})\mathbf{dx} = \mathbf{F}(\mathbf{b}) - \mathbf{F}(\mathbf{a})$$

Это основная формула интегрального исчисления (формула Ньютона–Лейбница). Практически все методы аналитического вычисления интегралов основаны на ней.

Почему бы ее не использовать всегда и для расчёта?

1.2 Уяснение ситуации.

1. Формулы для первообразной в классе элементарных функций может не быть:

$$\mathbf{f(x):}\quad \mathbf{e^{-x^2}},\quad \frac{\mathbf{\sin(x)}}{\mathbf{x}},\quad \frac{\mathbf{\cos(x)}}{\mathbf{x}},\quad \frac{\mathbf{1}}{\mathbf{\ln(x)}}$$

2. Даже, если интеграл берется в конечном виде, то методы интегрирования (поиска первообразной) нередко весьма трудоёмки.

3. Даже, если формула для первообразной и найдена, то расчет, выполненный на ЭВМ непосредственно по ней не всегда нас устроит. Например, в теоретических выкладках правой частью формулы

$$\int_{\mathbf{0}}^{\mathbf{1}} \mathbf{e^{-qx}dx} = \frac{\mathbf{1-e^{-q}}}{\mathbf{q}} \quad ,$$

являющейся аналитически точным результатом, можно пользоваться всегда. Однако (в случае недостаточной разрядости используемых переменных), программировать именно её для расчета на ЭВМ при $\mathbf{q \ll 1}$ плохо:

- при $\mathbf{q = 0}$ ЭВМ попытается выполнить деление на нуль;
- при $\mathbf{q = 10^{-8}}$ на арифметике одинарной точности ЭВМ получит $\mathbf{0}$ вместо $\mathbf{1}$, что заметить относительно легко;
- при $\mathbf{q = 10^{-7}}$ возникнет гораздо более опасная ситуация — результат окажется правдоподобен (**1.192093** вместо **.9999999**). Можно и не заметить, что в семизначной мантиссе **верна лишь одна** старшая значащая цифра.
- Наконец, значения подинтегральной функции могут получаться из эксперимента. Так что без дополнительных предположений о её свойствах формулы для первообразной в принципе не получить.

К счастью разработаны эффективные методы приближенного вычисления интегралов.

2 Основная формула приближённого интегрирования

Методы приближенного вычисления интегралов (или численного интегрирования) основаны на

- различных схемах разбиения промежутка интегрирования $[\mathbf{a}, \mathbf{b}]$ на подучастки;
- расчета вклада каждого участка в величину площади;
- суммировании вкладов всех участков.

В конечном итоге все эти методы сводятся к квадратурной формуле вида

$$\mathbf{S} \equiv \int_{\mathbf{a}}^{\mathbf{b}} \mathbf{f}(\mathbf{x})\mathbf{dx} = \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{C_i f}(\mathbf{x_i}) + \mathbf{R_f}(\mathbf{n})$$

Здесь $\mathbf{C_i}$ – веса квадратурной формулы; $\mathbf{x_i}$ – ее узлы; $\mathbf{R_f}(\mathbf{n})$ – остаток.

Если для заданного $\epsilon > \mathbf{0}$ можно найти такое $\mathbf{n}$, что

$$|\mathbf{R_f}(\mathbf{n})| \leq \epsilon \quad ,$$

то полагают, что квадратурная сумма

$$\tilde{\mathbf{S}} \equiv \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{C_i f}(\mathbf{x_i})$$

дает приближенную величину интеграла с абсолютной погрешностью, не превосходящей ϵ, то есть

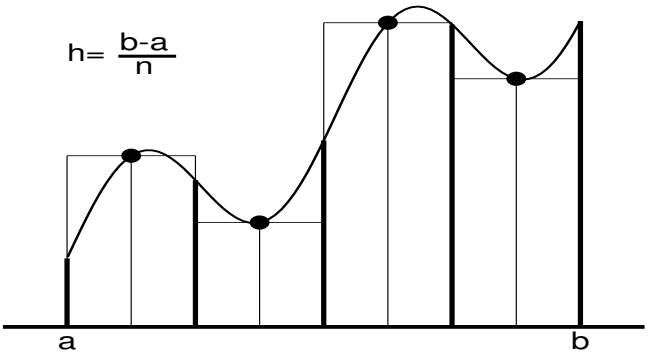
$$|\mathbf{S} - \tilde{\mathbf{S}}| \leq \epsilon$$

Таким образом, самое главное при использовании квадратурной формулы – это грамотное оценивание остатка $\mathbf{R_f}(\mathbf{n})$. Заметим, что остаток зависит не только от количества узлов $\mathbf{n}$, но и от вида подинтегральной функции $\mathbf{f}$. Это означает, что для разных функций квадратурная формула при одинаковом $\mathbf{n}$ имеет разную точность.

Из огромного множества квадратурных формул наиболее часто используются:

1. Формула средних прямоугольников;
2. Формула трапеций;
3. Формула Симпсона;
4. Формула Гаусса.

2.1 Формула средних прямоугольников.



$$\mathbf{S} \equiv \int_{\mathbf{a}}^{\mathbf{b}} \mathbf{f(x)dx}$$

$\mathbf{n}$ – количество участков дробления промежутка $[\mathbf{a}, \mathbf{b}]$.

$$\mathbf{h} = \frac{\mathbf{b}-\mathbf{a}}{\mathbf{n}}; \qquad \mathbf{x_i} = \mathbf{a} + \mathbf{h} \cdot (\mathbf{i} - \frac{\mathbf{1}}{\mathbf{2}}); \qquad \mathbf{i} = \mathbf{1}, \mathbf{2}, \cdots \mathbf{n}.$$

$$\mathbf{S} = \tilde{\mathbf{S}} + \mathbf{R_f(n)}$$

$$\tilde{\mathbf{S}} = \mathbf{h} \cdot \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{f(x_i)}$$

$$\mathbf{R_f(n)} = \frac{\mathbf{(b-a)^3}}{\mathbf{24n^2}} \cdot \mathbf{f''}(\xi).$$

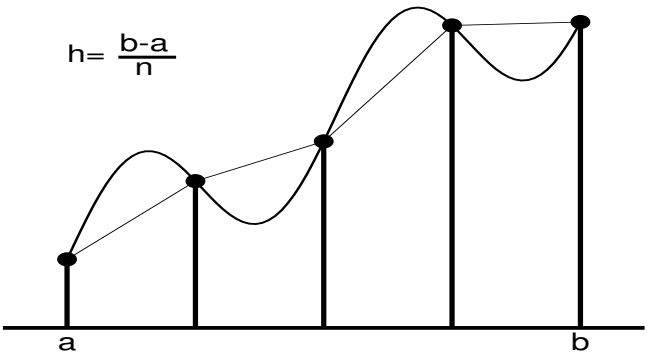
$\mathbf{a}$ и $\mathbf{b}$ – известны, $\mathbf{n}$ – задаем, но про ξ известно только одно, а именно, что $\xi \in [\mathbf{a}, \mathbf{b}]$.

Таким образом, если $|\mathbf{f''(x)}|$ ограничен на промежутке $[\mathbf{a}, \mathbf{b}]$ некоторым конечным и известным числом $\mathbf{M}$, то погрешность ограничения квадратурной суммы $\tilde{\mathbf{S}}$ просто оценить модулем мажоранты остатка $\mathbf{R_f(n)}$. И, наоборот, если задаться каким-то $\mathbf{eps} > \mathbf{0}$, и потребовать выполнения неравенства $|\mathbf{R_f(n)}| \leq \mathbf{eps}$, то по формуле

$$\mathbf{n} \geq \sqrt{\frac{\mathbf{(b-a)^3}}{\mathbf{24eps}} \cdot \mathbf{M}}$$

можно оценить минимальное $\mathbf{n}$, при котором квадратурная сумма обеспечит требуемую погрешность ограничения.

2.2 Формула трапеций.



$$\mathbf{S} \equiv \int_{\mathbf{a}}^{\mathbf{b}} \mathbf{f}(\mathbf{x})\mathbf{dx}.$$

$\mathbf{n}$ – количество участков дробления промежутка $[\mathbf{a}, \mathbf{b}]$.

$\mathbf{a} \equiv \mathbf{x_1}$; $\mathbf{b} \equiv \mathbf{x_{n+1}}$

$$\mathbf{h} = \frac{\mathbf{b}-\mathbf{a}}{\mathbf{n}}; \qquad \mathbf{x_i} = \mathbf{a} + \mathbf{h}\cdot(\mathbf{i}-\mathbf{1}); \qquad \mathbf{i} = \mathbf{1}, \mathbf{2}, \cdots \mathbf{n}+\mathbf{1}.$$

$$\mathbf{S} = \tilde{\mathbf{S}} + \mathbf{R_f}(\mathbf{n})$$

$$\tilde{\mathbf{S}} = \mathbf{h}\cdot\left[\frac{\mathbf{f}(\mathbf{a})+\mathbf{f}(\mathbf{b})}{\mathbf{2}}\right] + \sum_{\mathbf{i=2}}^{\mathbf{n}} \mathbf{f}(\mathbf{x_i})$$

$$\mathbf{R_f}(\mathbf{n}) = -\frac{(\mathbf{b}-\mathbf{a})^{\mathbf{3}}}{\mathbf{12n^2}}\cdot \mathbf{f}''(\xi).$$

И опять, если $|\mathbf{f}''(\mathbf{x})|$ ограничен на $[\mathbf{a}, \mathbf{b}]$, то по формуле

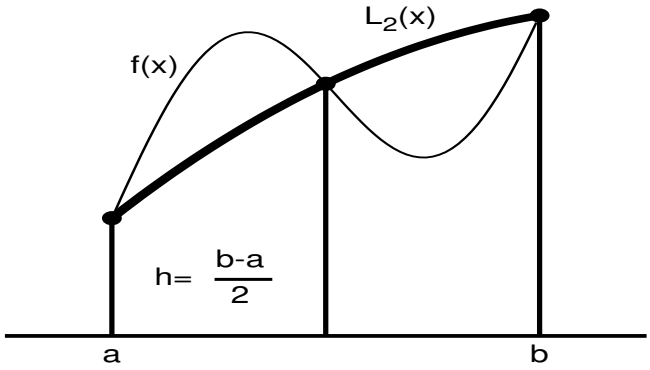
$$\mathbf{n} \geq \sqrt{\frac{(\mathbf{b}-\mathbf{a})^{\mathbf{3}}}{\mathbf{12eps}}\cdot \mathbf{M}}$$

можно оценить минимальное n, при котором квадратурная сумма обеспечит требуемую погрешность ограничения.

Замечание. Погрешность ограничения формулы средних прямоугольников вдвое меньше погрешности формулы трапеций.

2.3 Формула Симпсона.

В простейшем случае геометрическая иллюстрация формулы Симпсона имеет вид



где кривая $\mathbf{L_2(x)}$ — аппроксимирующий подинтегральную функцию интерполяционный полином второго порядка, т.е. парабола, проходящая через точки плоскости с координатами $(\mathbf{a}, \mathbf{f(a)})$, $(\mathbf{b}, \mathbf{f(b)})$, и $(\mathbf{a+h}, \mathbf{f(a+h)})$ (часто формулу Симпсона называют формулой парабол). Предполагается, что формула парабол позволит точнее вычислить площадь под кривой нежели формулы прямоугольников и трапеций.

Квадратурную сумму формулы Симпсона, т.е. результат интегрирования $\mathbf{L_2(x)}$ по промежутку $[a, b]$) при двух равных подучастках его дробления, можно записать в виде

$$\int_{\mathbf{a}}^{\mathbf{b}} \mathbf{f(x)dx} \approx \tilde{\mathbf{S}} = \frac{\mathbf{h}}{\mathbf{6}}\Big[\mathbf{f(a)} + \mathbf{4f}\Big(\frac{\mathbf{a+b}}{\mathbf{2}}\Big) + \mathbf{f(b)}\Big] \quad .$$

Наиболее широко используется составная формула Симпсона, основанная на дроблении промежутка интегрирования на четное число $\mathbf{n = 2m}$ равных подучастков.

$$\mathbf{n = 2 \cdot m} \quad ; \quad \mathbf{x_i = a + h \cdot (i - 1)} \quad , \quad \mathbf{i = 1, 2, \cdots, (n = 2m), n + 1.}$$

$$\mathbf{S = \tilde{S} + R_f(n)} \quad ; \quad \mathbf{\tilde{S} = \frac{h}{3} \cdot [f(a) + f(b) + 4 \cdot S_1 + 2 \cdot S_2]}$$

$\mathbf{S_1 = f_2 + f_4 + \cdots + f_{n=2m}}$; $\mathbf{f_i = f(x_i)}$;
$\mathbf{S_2 = f_3 + f_5 + \cdots + f_{n-1}}$.

$$\mathbf{R_f(n) = -\frac{(b-a)^5}{2880m^4} \cdot f^{(4)}(\xi); \quad \xi \in [a, b].}$$

Если $\mathbf{f(x)}$ четырежды непрерывно дифференцируема, т.е. $|\mathbf{f}^{(4)}|$ ограничен на $[\mathbf{a}, \mathbf{b}]$ числом $\mathbf{M}$, то можно для заданного ϵ найти величину $\mathbf{n = 2m}$, гарантирующую достижение требуемой погрешности метода:

$$\mathbf{n} \geq \sqrt[4]{\frac{(\mathbf{b-a})^5}{\mathbf{2880}\epsilon} \cdot \mathbf{M}}.$$

2.4 Формула Гаусса.

В классической форме имеет вид

$$\int_{-1}^{1} \mathbf{f(t)dt} = \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{C_i f(t_i)} + \mathbf{R_f(n)}.$$

Формула Гаусса основана на неравномерном дроблении промежутка интегрирования $[-\mathbf{1}, \mathbf{1}]$, которое способно обеспечить получение точного результата для всех полиномов степени не большей $\mathbf{2n-1}$. Так, при $\mathbf{n=3}$ квадратурная сумма формулы Гаусса гарантирует правильность результата численного интегрирования (с точностью до вычислительной погрешности) по промежутку $[-\mathbf{1}, \mathbf{1}]$ для любого полинома степени не большей $\mathbf{5}$, причём узлы $\mathbf{t_i}$ и веса $\mathbf{C_i}$ можно принять равными (в случае типа $REAL(4)$)

i	1	2	3
t_i	-0,7745667	0	+7745667
C_i	5.0/9.0	8.0/9.0	5.0/9.0

В общем случае промежутка $[\mathbf{a}, \mathbf{b}]$ имеем

$$\tilde{\mathbf{S}} = \int_{\mathbf{a}}^{\mathbf{b}} \mathbf{f(x)dx} = \frac{\mathbf{b-a}}{\mathbf{2}} \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{C_i} \cdot \mathbf{f}\Big(\frac{\mathbf{b-a}}{\mathbf{2}}\mathbf{t_i} + \frac{\mathbf{b+a}}{\mathbf{2}}\Big) \quad ;$$

$$\mathbf{R_f(n)} = \frac{\mathbf{(b-a)^{2n+1}(n!)^4}}{\mathbf{(2n!)^3(2n+1)}} \cdot \mathbf{f^{(2n)}}(\xi) \qquad \xi \in [\mathbf{a}, \mathbf{b}].$$

Замечания Формулы гауссова типа:

1. Обладают высокой точностью при небольшом числе узлов.
2. Имеют простой алгоритм суммирования.
3. Требуют помнить в оперативной памяти вектора узлов и весов, значения которых для заданного $\mathbf{n}$ можно взять из специальных таблиц (см., например, [9, 10]) или вычислить.
4. Существуют очень эффективные алгоритмы их расчета (например, [11, 6, 7]
5. Приведённые выше формулы Гаусса вычисляют интеграл в случае постоянной весовой функции. Однако нередко в подинтегральную функцию в качестве сомножителя входит некоторая весовая функция (например, $\mathbf{x}^{\alpha}$, $\mathbf{x}^{\alpha} \cdot (\mathbf{1} - \mathbf{x}^{\alpha})$, $\mathbf{x}^{\alpha} \cdot \mathbf{e}^{-\mathbf{x}}$ и др). Полезно знать, что и для них существуют соответствующие таблицы узлов и весов [9, 10] и алгоритмы быстрого расчёта [11, 6, 7].

3 Метод двойного пересчета.

Во многих задачах подинтегральные функции настолько сложны, что оценка остатка квадратурной формулы весьма трудоемка. Поэтому на практике часто используют доведение величины интеграла до заданной точности посредством метода двойного пересчета (правило Рунге): именно, перерасчет интеграла с уменьшением вдвое шага интегрирования до тех пор, пока разность между двумя последовательно полученными значениями интеграла не окажется достаточно мала.

Пусть $\tilde{\mathbf{S}}_{\mathbf{n}}$ квадратурная формулы при постоянном шаге дробления промежутка интегрирования

$$\tilde{\mathbf{S}}_{\mathbf{n}} = \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{C_i f}(\mathbf{x_i}) \quad .$$

Здесь $\mathbf{n} = \mathbf{2^k}$ – количество узлов. Построим последовательность $\{\tilde{\mathbf{S}}_{\mathbf{2^k}}\}$:

k	1	2	3	4	$\cdots$	p	$p+1$
$\tilde{S}_{2^k}$	$\tilde{S}_2$	$\tilde{S}_4$	$\tilde{S}_8$	$\tilde{S}_{16}$	$\cdots$	$\tilde{S}_{2^p}$	$\tilde{S}_{2^{p+1}}$

Допустим, что при $\mathbf{k} \to \infty$ $\tilde{\mathbf{S}}_{\mathbf{2^k}}$ стремиться к истинной величине интеграла.

Зададимся некоторым $\epsilon > \mathbf{0}$ и прекратим вычисление $\tilde{\mathbf{S}}_{\mathbf{2^k}}$ как только окажется, что при очередном удвоении количества узлов

$$|\tilde{\mathbf{S}}_{\mathbf{2^p}} - \tilde{\mathbf{S}}_{\mathbf{2^{p+1}}}| \leq \epsilon \quad ,$$

после чего в качестве основного результата (величины искомого интеграла $\mathbf{S}$ примем значение $\mathbf{S}_{\mathbf{2^{p+1}}}$. Строго говоря, в общем случае, нельзя гарантировать, что

$$|\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{2^{p+1}}}| \leq \epsilon \quad .$$

Однако, если предположить, что поведение остатка квадратурной формулы такое же как у функции $\mathbf{h^m}$, где $\mathbf{m} \geq \mathbf{1}$ или, другими словами,

$$\mathbf{R_f(n)} = \mathbf{O(h^m)} \quad ,$$

или, что то же

$$\mathbf{R_f(n)} = \lim_{\substack{\mathbf{n} \to \infty \\ \mathbf{h} \to \mathbf{0}}} \frac{\mathbf{R_f(n)}}{\mathbf{h^m}} = \mathbf{const} \quad ,$$

или, иначе говоря,

$$\mathbf{R_f(n)} = \mathbf{M} \cdot \mathbf{h^m} \quad ,$$

то будем иметь

$$\frac{\mathbf{R_f(n1)}}{\mathbf{R_f(n2)}} = \frac{\mathbf{M} \cdot \mathbf{h_{n1}^m}}{\mathbf{M} \cdot \mathbf{h_{n2}^m}} = \frac{\left(\dfrac{\mathbf{b-a}}{\mathbf{n1}}\right)^{\mathbf{m}}}{\left(\dfrac{\mathbf{b-a}}{\mathbf{n2}}\right)^{\mathbf{m}}} = \left(\frac{\mathbf{n2}}{\mathbf{n1}}\right)^{\mathbf{m}} = \frac{\mathbf{1}}{\alpha^{\mathbf{m}}} \quad .$$

Здесь $\alpha \equiv \dfrac{\mathbf{n1}}{\mathbf{n2}} < \mathbf{1}$ – отношение количества узлов квадратуры до и после уменьшения шага. Заметим, что пока правило уменьшения шага (или что по сути – то же, правило увеличения количества узлов с **n1** до **n2**) не используется. Важно лишь, чтобы оно приводило к справедливости неравенства $\mathbf{n1} < \mathbf{n2}$. Тогда

$$\frac{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n1}}}{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}}} = \frac{\mathbf{R_f(n1)}}{\mathbf{R_f(n2)}} = \frac{\mathbf{1}}{\alpha^{\mathbf{m}}} \quad ,$$

то есть отличие текущего приближения от истинной величины интеграла больше соответствующего отличия следующего (поскольку $\dfrac{\mathbf{1}}{\alpha^{\mathbf{m}}} > \mathbf{1}$). Таким образом, если надеемся, что остаток квадратурной формулы пропорционален $\mathbf{h^m}$, то приближение значения квадратурной суммы к правильному результату гарантируется. Поскольку правильный результат – неизвестен, то полезно выяснить: "Как отличие двух последовательных приближений связано с отличием окончательного приближения от точного значения?"

$$\frac{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n1}}}{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}}} = \frac{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}} + \tilde{\mathbf{S}}_{\mathbf{n2}} - \tilde{\mathbf{S}}_{\mathbf{n1}}}{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}}} = \mathbf{1} + \frac{\tilde{\mathbf{S}}_{\mathbf{n2}} - \tilde{\mathbf{S}}_{\mathbf{n1}}}{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}}} = \frac{\mathbf{1}}{\alpha^{\mathbf{m}}} \quad .$$

Так что

$$\frac{\tilde{\mathbf{S}}_{\mathbf{n2}} - \tilde{\mathbf{S}}_{\mathbf{n1}}}{\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}}} = \frac{\mathbf{1}}{\alpha^{\mathbf{m}}} - \mathbf{1} = \frac{\mathbf{1} - \alpha^{\mathbf{m}}}{\alpha^{\mathbf{m}}} \quad .$$

И, окончательно,

$$|\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}}| = \frac{\alpha^{\mathbf{m}}}{\mathbf{1} - \alpha^{\mathbf{m}}} \cdot |\tilde{\mathbf{S}}_{\mathbf{n2}} - \tilde{\mathbf{S}}_{\mathbf{n1}}| \quad .$$

Это означает, что абсолютная погрешность метода двойного пересчета $|\mathbf{S} - \tilde{\mathbf{S}}_{\mathbf{n2}}|$ будет составлять от требуемого ϵ (разности двух последовательных приближений) не более доли $\dfrac{\alpha^{\mathbf{m}}}{\mathbf{1} - \alpha^{\mathbf{m}}}$, то есть окажется меньше ϵ в $\dfrac{\mathbf{1} - \alpha^{\mathbf{m}}}{\alpha^{\mathbf{m}}}$ раз.
В частности, при двукратном увеличении количества узлов ($\alpha = \mathbf{0.5}$) имеем:

Формула	$\mathbf{R_f(n)}$	$\mathbf{m}$	$\dfrac{\mathbf{1} - \alpha^{\mathbf{m}}}{\alpha^{\mathbf{m}}}$
средних прямоугольников	$\mathbf{O(h^2)}$	**2**	**3**
трапеций	$\mathbf{O(h^2)}$	**2**	**3**
Симпсона	$\mathbf{O(h^4)}$	**4**	**15**

Из таблицы видно, что, если добились между двумя последовательными приближениями метода двойного пересчета отличия меньшего, например, $\mathbf{0,01}$, то реально достигнутая точность в случае метода Симпсона в **15** раз выше.

Замечание Метод двойного пересчёта часто называют методом Рунге.

4 Информация к размышлению.

1. Формулы оценки остатка квадратурной формулы позволяют выделить класс подинтегральных функций, для которого квадратурная сумма должна дать точный результат (в пределах погрешности округления).

2. В случае формул средних прямоугольников и трапеций к такому классу функций относятся все полиномы степени не выше первой (*Почему?*).

3. Процедура расчёта по формуле средних прямоугольников, в классе линейных функций должна получать точный результат даже при использовании **одного** узла.

4. Процедура расчёта по формуле трапеций в классе линейных функций, должна получать точный результат и по двум узлам, и по трем узлам. Трёхузловой тест подтвердит правильность работы циклической части алгоритма, а двухузловой — правильность учёта вклада от концевых узлов.

5. В случае формулы Симпсона подпрограмма должна давать верный результат для всех полиномов степени не выше третьей (*Почему?*). В тестовой программе полезно убедиться в правильности работы процедуры для функций $1, x, x^2, x^3$ при двух, четырех, и шести шагов дробления промежутка интегрирования.

6. Суммирование в формуле Симпсона можно элегантно записать одним оператором цикла. Опыт свидетельствует, что начальные попытки реализации подобной схемы чреваты ошибками, психологически незаметными на первый взгляд.

7. Помним, что при использовании формулы Симпсона количество промежутков дробления должно быть обязательно четным.

8. Помним, что результат суммирования формул прямоугольников, трапеций и Сипсона нужно умножать на шаг (это иногда забывается).

9. В литературе нумерация узлов формул интегрирования с постоянным шагом иногда начинается с 0, а иногда с 1. В первом случае номер последнего узла равен числу промежутков дробления, а во вторых — на единицу больше (что выгоднее для программирования — решать нам).

10. Не следует отлаживать или тестировать процедуру на проблемной подинтегральной функции. Ошибочный учёт вклада в величину интеграла (например, правой концевой точки) может маскироваться пренебрежимо малым значением подинтегральной функции. Таким образом, возникает опасная иллюзия о правильной работе процедуры. Именно поэтому важно тестирование, которое получает верный результат при минимальном числе узлов.

11. Предварительное тестирование важно и для процедур, входящих в различные математические пакеты. Испытание их работы на тестовых примерах позволит не допустить досадных промахов при написании проблемных программ.

4.1 Выбор способа задания подинтегральной функции.

1. Перед оформлением алгоритма численного интегрирования процедурой полезно уяснить оптимальный способ задания подинтегральной функции: аналитический или табличный. Способ задания зависит от специфики задачи.

2. Аналитическое задание позволяет вычислять подинтегральную функцию в любой точке промежутка интегрирования и допускает расчет интеграла с автоматическим выбором шага. В то же время оно не исключает переход на табличный способ задания, если последний покажется более удобным.

3. Табличное задание предполагает расчет только квадратурной суммы по набору известных значений подинтегральной функции, которые получены либо из эксперимента, либо из расчета по формуле. Так что оценка остатка квадратурной формулы на совести пользователя.

4. Иногда область интегрирования такова, что подинтегральная функция имеет в ней быстро сходящееся разложение в ряд Тейлора, которое можно проинтегрировать аналитически и найти соответствующий ряд для искомого интеграла. Оценка остатка полученного ряда зачастую гораздо проще оценки остатка квадратурной формулы. Таким образом, оказывается возможным независимый расчет, а значит и дополнительный контроль правильности.

5. Табличное задание иногда выгодно и при наличии аналитической формулы. Например, пусть подинтегральная функция $\mathbf{f}(\mathbf{p},\mathbf{x}) = \mathbf{e}^{-\mathbf{px}} \cdot \mathbf{g}(\mathbf{x})$ зависит от параметра $\mathbf{p}$ и требует много времени на расчет входящей в нее $\mathbf{g}(\mathbf{x})$ (по сравнению со временем расчета $\mathbf{e}^{-\mathbf{px}}$). Пусть необходимо вычислить интеграл от $\mathbf{f}(\mathbf{p},\mathbf{x})$ для большого количества значений параметра $\mathbf{p}$ из приемлемого диапазона. Если в узлах квадратурной формулы вычислить набор значений $\mathbf{g}(\mathbf{x})$ и запомнить его в элементах вектора, то можно избежать многократного расчета одних и тех же $\mathbf{g}(\mathbf{x})$ для разных $\mathbf{p}$.

   Для получения значений $\mathbf{g}(\mathbf{x})$ в неузловых точках (если таковые потребуются) можно использовать ту или иную процедуру интерполяции, время работы которой значительно меньше временных затрат на непосредственный расчёт $\mathbf{g}(\mathbf{x})$.

5 Расчёт узлов и весов квадратур гауссового типа

В этом разделе приведёны процедуры расчёта узлов и весов нескольких квадратур гауссового типа

1) **fqjac0**: для единичной весовой функции для промежутков [-1,1] или [0,1].

2) **fqlag0**: с весовой функцией $\mathbf{e^{-x}}$ по полуоси $[0, \infty)$.

3) **fqlag**: для полуоси с весовой функцией $\mathbf{x^{\alpha} e^{-x}}$.

4) **gauleg**: для единичной весовой функции по промежутку [a,b]

5) **gaulag**: с весовой функцией $\mathbf{x^{\alpha} e^{-x}}$ по полуоси $[0, \infty)$.

6) **gauerm**: с весовой функцией e^{-x^2} на интервале $-\infty, +\infty$.

7) **gaujac** с весовой функцией $(1-x)^{\alpha}(1+x)^{\beta}$ по промежутку [-1,1]

и тестирующие их программы. Исходные тексты первых трёх процедур написаны в стиле древнего ФОРТРАНа. Тем не менее, приводимые тексты должны проходить и на современных компиляторах. Последние четыре представляют собой слегка изменённые версии подпрограмм **gauleg**, **gaulag gauerm** и **gaujac** соответствующих подпрограмм из [7].

В [7] список формальных аргументов минимален — не содержит количество вычисляемых узлов, поскольку современный ФОРТРАН позволяет определить это количество посредством встроенной ФОРТРАН-функции **size**, что является одновременно и достоинством, и недостатком.

- Достоинство очевидно — чем меньше аргументов, тем меньше вероятность допущения ошибки при вызове.
- Недостаток: невозможность без существенной переработки старых программ указать при вызове алгоритмов расчёта гауссовых квадратур нужное количество их узлов. В старых программах нередко один массив (описанный в пределах разумного максимума) мог обслуживать квадратуры с разным числом узлов, которое передавалось в качестве фактического аргумента. Функция же **size** всегда укажет заявленный размер массива (т.е. максимальный), что (в случае старых программ) неудобно.

Тестирование процедур **1)-6)** состоит в выводе абсолютной и относительной погрешностей моментов полинома $\mathbf{x^m}$ ($\mathbf{m = 0(1)2k-1; k}$ — число узлов квадратуры), если за точный результат принять значение, получаемое по формуле Ньютона-Лейбница, а за приближённый — посредством квадратур. Как известно, когда подинтегральная функция является полиномом степени не выше **(2k-1)**, квадратуры Гаусса при положительной весовой функции формально должны давать точный результат (см., например, [9, 10]). В случае же квадратуры **gaujac** в качестве контролирующего подинтегрального выражения использовался квадрат полинома Якоби $\mathbf{P_m^{(\alpha,\beta)}}$ при весовой функции $\mathbf{(1-x)^{\alpha} \cdot (1+x)^{\beta}}$.

5.1 Подпрограмма fqjac0 и её тестирование

```
      program test_fqjac0; implicit none
      real(8) t(1000), a(1000), eps, c0, aa1, aa2, za, zg, d, r
      real(8) aa(-1:0) /-1d0,0d0/
      character(60) :: txt(-1:0)=(/'Промежуток интегрирования (-1,1)',
     >                             'Промежуток интегрирования (0, 1)'/)
      integer n, k, i, j
      read (*,'(i5)') n, k;   read (*, '(d10.3)') eps
      write(*,*) trim(txt(n))
      write (*, '(" Кол-во вычисляемых узлов  ( k )=",i5)') k
      write (*, '(" Отн. погрешность узлов    (eps)=",d10.3)') ,eps
      call fqjac0(n,k,eps,t,a)
      write (*, '("    i",12x,"Узлы",21x,"Веса")')
      write (*, 15) (i,t(i),a(i),i=1,k)
      write (*, 16) 'j','По Ньютону-Лейбницу','По Гауссу','абс','отн'
      c0=0d0
      aa1=aa(n)
      aa2=1
      do j=0,2*k-1
        za=(aa2**(j+1))/(j+1)-(aa1**(j+1))/(j+1) ! Ньютон-Лейбниц
        zg=0
        do i=1,k
          zg=zg+a(i)*t(i)**j
        enddo
        d=abs(za-zg)
        r=dabs(d/za)
        write (*, 17) j,za,zg,d,r
      enddo
   15 format(1x,i4,2x,d23.15,2x,d23.15)
   16 format(//4x,a,4x,a,10x,a,11x,a,8x,a/)
   17 format(1x,i4,1x,d23.15,1x,d23.15,2x,d10.4,1x,d10.4)
      end
```

```
!=====================================================================
! Подпрограмма fqjac0(n,k,eps,t,a) вычисляет узлы и веса k-точечной
! квадратуры Гаусса при n=-1 для промежутка [-1,1], иначе --- для [0,1].
! eps - относительная погрешность расчёта узлов.
! Первые (k) элементов векторов (t) и (a) содержат после работы
! процедуры вычисленные значения узлов и весов соответственно.
! Алгоритм представляет переложение на язык ФОРТРАН соответствующей
! Алгол-процедуры взятой из АЛГОЛ-ПРОЦЕДУРЫ, ВЫПУСК 12, СТР.6;
! А.К.Пономаренко. Изд. Ленгос. Ун-та  1974г.
! .....................................................................
      SUBROUTINE FQJAC0(N,K,EPS,T,A)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION T(K),A(K)
      DATA C1,C2,C3,C41,C8,C246 / 1D0, 2D0, 3D0, 4.1D0, 8D0, 0.246D0 /
      DATA C05,C203,C292 / 0.5D0, 2.0374D0, 2.9225D0 /
      I  = K
      DK = K; DKD= C1/DK; X = C1-C2/(DK*DK+DK); GOTO 40
   10 CONTINUE
         TK=T(K); X=TK+(TK-C1)*(C41+C246*(DK-C8)*DKD); GOTO 40
   20 CONTINUE
         K1=K-1; TK1=T(K1); X=TK1+(TK1-T(K))*(C203-C292*DKD); GOTO 40
   30 CONTINUE
        X=(T(I+1)-T(I+2))*C3+T(I+3)
   40 CONTINUE
        D2=C1
        D3=X
        IF (K.NE.1) THEN
          DO 50 J=2,K
            D1=D2
            D2=D3
            DJ=J
            DJ1=DJ-C1
            D3=(X*(DJ+DJ1)*D2-DJ1*D1)/DJ
   50     CONTINUE
        ENDIF
        D1=DKD/(D2-X*D3)
        D2=D3*D1*(C1-X*X)
        X=X-D2
      IF (DABS(D2).GE.EPS) GOTO 40
      T(I)=X
      A(I)=D1*D1*(C1-X*X)
      IF (N.EQ.-1) A(I)=A(I)*C2
      I=I-1
      IF ( I.EQ.  0 ) GOTO 60
      IF ( I.EQ.K-2 ) GOTO 20
      IF ( I.EQ.K-1 ) GOTO 10
      GOTO 30
   60 CONTINUE
      IF (N.EQ.-1) RETURN
      DO 70 J=1,K
        T(J)=(C1+T(J))*C05
   70 CONTINUE
      RETURN
      END
```

Результаты работы **test_fqjac0** и **fqjac0** для $\mathbf{k = 10}$ и $\mathbf{eps = 10^{-10}}$:

```
Промежуток интегрирования (-1,1)
Кол-во вычисляемых узлов  ( k )=   10
Отн. погрешность узлов    (eps)= 0.100D-08
   i            Узлы                    Веса
   1   -0.973906528517172D+00    0.666713443086883D-01
   2   -0.865063366688985D+00    0.149451349150581D+00
   3   -0.679409568299024D+00    0.219086362515982D+00
   4   -0.433395394129247D+00    0.269266719309996D+00
   5   -0.148874338981631D+00    0.295524224714753D+00
   6    0.148874338981631D+00    0.295524224714753D+00
   7    0.433395394129247D+00    0.269266719309996D+00
   8    0.679409568299024D+00    0.219086362515982D+00
   9    0.865063366688985D+00    0.149451349150581D+00
  10    0.973906528517172D+00    0.666713443086880D-01


  j    По Ньютону-Лейбницу          По Гауссу           абс        отн

  0   0.200000000000000D+01   0.200000000000000D+01  0.0000D+00 0.0000D+00
  1   0.000000000000000D+00  -0.222044604925031D-15  0.2220D-15   Infinity
  2   0.666666666666667D+00   0.666666666666667D+00  0.1110D-15 0.1665D-15
  3   0.000000000000000D+00  -0.228983498828939D-15  0.2290D-15   Infinity
  4   0.400000000000000D+00   0.400000000000000D+00  0.5551D-16 0.1388D-15
  5   0.000000000000000D+00  -0.187350135405495D-15  0.1874D-15   Infinity
  6   0.285714285714286D+00   0.285714285714286D+00  0.0000D+00 0.0000D+00
  7   0.000000000000000D+00  -0.180411241501588D-15  0.1804D-15   Infinity
  8   0.222222222222222D+00   0.222222222222222D+00  0.0000D+00 0.0000D+00
  9   0.000000000000000D+00  -0.159594559789866D-15  0.1596D-15   Infinity
 10   0.181818181818182D+00   0.181818181818182D+00  0.0000D+00 0.0000D+00
 11   0.000000000000000D+00  -0.131838984174237D-15  0.1318D-15   Infinity
 12   0.153846153846154D+00   0.153846153846154D+00  0.0000D+00 0.0000D+00
 13   0.000000000000000D+00  -0.145716771982052D-15  0.1457D-15   Infinity
 14   0.133333333333333D+00   0.133333333333333D+00  0.0000D+00 0.0000D+00
 15   0.000000000000000D+00  -0.117961196366423D-15  0.1180D-15   Infinity
 16   0.117647058823529D+00   0.117647058823529D+00  0.1388D-16 0.1180D-15
 17   0.000000000000000D+00  -0.832667268468867D-16  0.8327D-16   Infinity
 18   0.105263157894737D+00   0.105263157894737D+00  0.0000D+00 0.0000D+00
 19   0.000000000000000D+00  -0.693889390390723D-16  0.6939D-16   Infinity
```

Замечания:

- Бесконечное значение относительной погрешности, когда показатель степени интегрируемого полинома **нечётен** не должно волновать, так как точное значение интеграла по симметричному промежутку **[-1,1]** равно **нулю**. Результат же численного интегрирования в пределах арифметики **real(8)** оказался не хуже $\mathbf{10^{-15}}$ (см. столбец с абсолютной погрешностью), что более чем приемлемо (ведь точность расчёта узлов квадратуры была всё-таки $\mathbf{10^{-9}}$).

- Для промежутка **[-1,1]** узлы и веса обладают следующей симметрией относительно t=0: $\mathbf{t(n+1-k) = -t(k)}$, $\mathbf{A(n+1-k) = A(k)}$.

- Тестирование **fqjac0** по промежутку **[0,1]** (**n=0**) привело к результату:

```
Промежуток интегрирования (0, 1)
Кол-во вычисляемых узлов  ( k )=   10
Отн. погрешность узлов    (eps)= 0.100D-08
   i           Узлы                    Веса
   1    0.130467357414142D-01    0.333356721543441D-01
   2    0.674683166555077D-01    0.747256745752903D-01
   3    0.160295215850488D+00    0.109543181257991D+00
   4    0.283302302935376D+00    0.134633359654998D+00
   5    0.425562830509184D+00    0.147762112357376D+00
   6    0.574437169490816D+00    0.147762112357376D+00
   7    0.716697697064624D+00    0.134633359654998D+00
   8    0.839704784149512D+00    0.109543181257991D+00
   9    0.932531683344492D+00    0.747256745752903D-01
  10    0.986953264258586D+00    0.333356721543440D-01


   j    По Ньютону-Лейбницу          По Гауссу           абс        отн

   0   0.100000000000000D+01   0.100000000000000D+01  0.0000D+00 0.0000D+00
   1   0.500000000000000D+00   0.500000000000000D+00  0.5551D-16 0.1110D-15
   2   0.333333333333333D+00   0.333333333333333D+00  0.5551D-16 0.1665D-15
   3   0.250000000000000D+00   0.250000000000000D+00  0.5551D-16 0.2220D-15
   4   0.200000000000000D+00   0.200000000000000D+00  0.5551D-16 0.2776D-15
   5   0.166666666666667D+00   0.166666666666667D+00  0.2776D-16 0.1665D-15
   6   0.142857142857143D+00   0.142857142857143D+00  0.0000D+00 0.0000D+00
   7   0.125000000000000D+00   0.125000000000000D+00  0.2776D-16 0.2220D-15
   8   0.111111111111111D+00   0.111111111111111D+00  0.0000D+00 0.0000D+00
   9   0.100000000000000D+00   0.100000000000000D+00  0.1388D-16 0.1388D-15
  10   0.909090909090909D-01   0.909090909090909D-01  0.0000D+00 0.0000D+00
  11   0.833333333333333D-01   0.833333333333333D-01  0.0000D+00 0.0000D+00
  12   0.769230769230769D-01   0.769230769230769D-01  0.0000D+00 0.0000D+00
  13   0.714285714285714D-01   0.714285714285714D-01  0.0000D+00 0.0000D+00
  14   0.666666666666667D-01   0.666666666666667D-01  0.1388D-16 0.2082D-15
  15   0.625000000000000D-01   0.625000000000000D-01  0.1388D-16 0.2220D-15
  16   0.588235294117647D-01   0.588235294117647D-01  0.0000D+00 0.0000D+00
  17   0.555555555555556D-01   0.555555555555556D-01  0.0000D+00 0.0000D+00
  18   0.526315789473684D-01   0.526315789473684D-01  0.0000D+00 0.0000D+00
  19   0.500000000000000D-01   0.500000000000000D-01  0.6939D-17 0.1388D-15
```

Замечание. Современные версии **gfortran** имеют, в частности, опции:

-fdefault-real-8, **-freal-4-real-10**, **-freal-4-real-16**,
-freal-4-real-8, **-freal-8-real-10**, **-freal-8-real-16**,
-freal-8-real-4,

которые обеспечивают переключение одного режима описания переменных вещественного типа на другой. Например, программа **test_fqjac0** после компиляции с опцией **-freal-8-real-10** получит даже при **eps=1e-11** абсолютную погрешность интегрального контроля не хуже $\sim \mathbf{10^{-18}}$:

```
Промежуток интегрирования (-1,1)
Кол-во вычисляемых узлов  ( k )=   10
Отн. погрешность узлов    (eps)= 0.100D-10
   i            Узлы                    Веса
   1   -0.973906528517172D+00    0.666713443086881D-01
   2   -0.865063366688985D+00    0.149451349150581D+00
   3   -0.679409568299024D+00    0.219086362515982D+00
   4   -0.433395394129247D+00    0.269266719309996D+00
   5   -0.148874338981631D+00    0.295524224714753D+00
   6    0.148874338981631D+00    0.295524224714753D+00
   7    0.433395394129247D+00    0.269266719309996D+00
   8    0.679409568299024D+00    0.219086362515982D+00
   9    0.865063366688985D+00    0.149451349150581D+00
  10    0.973906528517172D+00    0.666713443086881D-01


   j    По Ньютону-Лейбницу          По Гауссу           абс        отн

   0   0.200000000000000D+01   0.200000000000000D+01  0.3253D-18 0.1626D-18
   1   0.000000000000000D+00  -0.101643953670516D-18  0.1016D-18 -.1000D+01
   2   0.666666666666667D+00   0.666666666666667D+00  0.1084D-18 0.1626D-18
   3   0.000000000000000D+00  -0.711507675693612D-19  0.7115D-19 -.1000D+01
   4   0.400000000000000D+00   0.400000000000000D+00  0.2711D-19 0.6776D-19
   5   0.000000000000000D+00  -0.474338450462408D-19  0.4743D-19 -.1000D+01
   6   0.285714285714286D+00   0.285714285714286D+00  0.0000D+00 0.0000D+00
   7   0.000000000000000D+00  -0.542101086242752D-19  0.5421D-19 -.1000D+01
   8   0.222222222222222D+00   0.222222222222222D+00  0.1355D-19 0.6099D-19
   9   0.000000000000000D+00  -0.542101086242752D-19  0.5421D-19 -.1000D+01
  10   0.181818181818182D+00   0.181818181818182D+00  0.1355D-19 0.7454D-19
  11   0.000000000000000D+00  -0.474338450462408D-19  0.4743D-19 -.1000D+01
  12   0.153846153846154D+00   0.153846153846154D+00  0.1355D-19 0.8809D-19
  13   0.000000000000000D+00  -0.440457132572236D-19  0.4405D-19 -.1000D+01
  14   0.133333333333333D+00   0.133333333333333D+00  0.1355D-19 0.1016D-18
  15   0.000000000000000D+00  -0.406575814682064D-19  0.4066D-19 -.1000D+01
  16   0.117647058823529D+00   0.117647058823529D+00  0.2033D-19 0.1728D-18
  17   0.000000000000000D+00  -0.372694496791892D-19  0.3727D-19 -.1000D+01
  18   0.105263157894737D+00   0.105263157894737D+00  0.2033D-19 0.1931D-18
  19   0.000000000000000D+00  -0.372694496791892D-19  0.3727D-19 -.1000D+01
```

5.2 Подпрограмма fqlag0 и её тестирование

Как известно,

$$\int_0^\infty \mathbf{x^\alpha e^{-x} x^m dx} = \mathbf{\Gamma(1 + m + \alpha)}$$

Если значение **Γ**-функции вычисляется независимым способом, то интегральный контроль точности узлов и весов, найденных **fqlag0** и **fqlag**, можно осуществить посредством расчёта интегралов от функции $\mathbf{x^m}$ (**m=0(1)2k-1**; k — число узлов).

Для контроля **fqlag0** положим $\alpha = \mathbf{0}$. Тогда значение интеграла должно равняться **m!** при любом неотрицательном целом **m<2k**. Расчёт очередного значения факториала достигается домножением предыдущего его значения на очередное **m**.

Для контроля **fqlag** значение **alpha** должно быть больше **-1**. В этом случае для независимого расчёта **Γ**-функции придётся вызывать соответствующую встроенную ФОРТРАН-функцию **gamma**.

```
      program test_fqlag0
      implicit none
      real(8) a(1000), t(1000)
      real(8) eps, q, s
      integer k, i, m
      read (*,  * ) k, eps
      write(*,1000) k, eps
      call fqlag0( k, eps, t, a )
      write(*,1010)
      write(*,1100) (i, t(i),a(i),i=1,k )
      write(*,*)
      write(*,1001)
      q=1                          ! Текущее значение m!
      do m=0,(2*k-1)
         if (m.gt.0) q=m*q
         s=0
         do i=1,k
           s=s+a(i)*t(i)**(m)
         enddo
         Write(*,1011) m, s, q, abs(s-q), abs((s-q)/s)
      enddo
 1000 format(1x,'k= ',i3,3x,'eps= ',D23.16)
 1001 format(5x,'m',9x,'Численно',16x,'Точно'
     >,13x,'абс.',7x,'отн.')
 1010 format(5x,'i',14x,'t',23x,'a')
 1011 format(1x,i5,1x,d22.14,1x,d22.14,2x,d9.2,2x,d9.2)
 1100 format(1x,i5,3x,d22.17,3x,d22.17)
      end
```

```
c=====================================================================
c Подпрограмма вычисляет с погрешностью (eps) узлы и веса k-точечной
c квадратуры по полуоси с весовой функцией exp(-x).
c После работы подпрограммы первые (k) элементов вектора (t) хранят
c найденные узлы квадратуры, а первые (k) элементов вектора (a)
c хранят соответствующие веса.
c Алгоритм представляет переложение на язык ФОРТРАН соответствующей
c Алгол-процедуры взятой из АЛГОЛ-ПРОЦЕДУРЫ, ВЫПУСК 12, СТР.6;
c А.К.Пономаренко. Изд. Ленгос. Ун-та  1974г.
c.....................................................................
      SUBROUTINE FQLAG0(K,EPS,T,A)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION T(k),A(k)
      I=0
      DP=K
      X=3D0/(1D0+2.4D0*DP)
      GO TO 30
   10 X=T(I)+6D0/(0.4D0+DP)
      GO TO 30
   20 J=I-1
      DJ=J
      X=T(I)+(T(I)-T(J))*(1D0+2.55D0*DJ)/(1.9D0*DJ)
   30 E2=1D0
      E3=1D0-X
      IF(K.EQ.1) GO TO 41
      DO 40 J=2,K
         E1=E2
         E2=E3
         DJ=J
         DJ1=DJ-1D0
   40    E3=((DJ+DJ1-X)*E2-DJ1*E1)/DJ
   41    E1=DP*(E3-E2)
         E2=E3/E1
         X=X*(1D0-E2)
         IF ((DABS(E2)-EPS).ge.0d0) goto 30
         I=I+1
         T(I)=X
         A(I)=X/(E1*E1)
         if ((i-1) .eq.0) goto 10
         if ((i-k).lt.0) goto 20
         if ((i-k).gt.0) stop 66
         RETURN
         END
```

Результаты тестирования **fqlag0** при $\mathbf{k = 10}$ и $\mathbf{eps = 10^{-10}}$:

```
k=  10   eps=  0.9999999999999999D-11
    i             t                        a
    1   .13779347054049237D+00   .30844111576501998D+00
    2   .72945454950317035D+00   .40111992915527361D+00
    3   .18083429017403159D+01   .21806828761180946D+00
    4   .34014336978548991D+01   .62087456096856078D-01
    5   .55524961400638038D+01   .95015169751811231D-02
    6   .83301527467644974D+01   .75300838858749248D-03
    7   .11843785837900064D+02   .28259233495767001D-04
    8   .16279257831378104D+02   .42493139842536472D-06
    9   .21996585811980761D+02   .18395648235605021D-08
   10   .29920697012273894D+02   .99118272139817771D-12

    m        Численно                Точно              абс.       отн.
    0   0.99999999999818D+00   0.10000000000000D+01   0.18D-11   0.18D-11
    1   0.99999999999380D+00   0.10000000000000D+01   0.62D-11   0.62D-11
    2   0.19999999999789D+01   0.20000000000000D+01   0.21D-10   0.11D-10
    3   0.59999999999276D+01   0.60000000000000D+01   0.72D-10   0.12D-10
    4   0.23999999999746D+02   0.24000000000000D+02   0.25D-09   0.11D-10
    5   0.11999999999903D+03   0.12000000000000D+03   0.97D-09   0.81D-11
    6   0.71999999999517D+03   0.72000000000000D+03   0.48D-08   0.67D-11
    7   0.50399999999603D+04   0.50400000000000D+04   0.40D-07   0.79D-11
    8   0.40319999999505D+05   0.40320000000000D+05   0.50D-06   0.12D-10
    9   0.36287999999262D+06   0.36288000000000D+06   0.74D-05   0.20D-10
   10   0.36287999998830D+07   0.36288000000000D+07   0.12D-03   0.32D-10
   11   0.39916799998088D+08   0.39916800000000D+08   0.19D-02   0.48D-10
   12   0.47900159996802D+09   0.47900160000000D+09   0.32D-01   0.67D-10
   13   0.62270207994525D+10   0.62270208000000D+10   0.55D+00   0.88D-10
   14   0.87178291190384D+11   0.87178291200000D+11   0.96D+01   0.11D-09
   15   0.13076743678261D+13   0.13076743680000D+13   0.17D+03   0.13D-09
   16   0.20922789884749D+14   0.20922789888000D+14   0.33D+04   0.16D-09
   17   0.35568742803287D+15   0.35568742809600D+15   0.63D+05   0.18D-09
   18   0.64023737044490D+16   0.64023737057280D+16   0.13D+07   0.20D-09
   19   0.12164510038171D+18   0.12164510040883D+18   0.27D+08   0.22D-09
```

5.3 Подпрограмма fqlag и её тестирование

```
! ======================================================================
! Программа посредством численного интегрирования функций
! f(x)=x**m (m=0 (1) 2k-1) k-точечной квадратурой Гаусса-Лаггера вида
!  [0,...) x^al exp(-x) f(x) dx = [n=1,k] A(n) f(x(n)) = Г(al+m+1),
! где Г(x) - Гамма-функция, оценивает точность расчёта узлов и весов
! квадратуры. Если они верны, то квадратурная сумма с точностью до
! погрешности их расчёта должна совпасть со значением Г(al+m+1).
! ......................................................................
      implicit real*8(A-H,O-Z)
      dimension A(1000), T(1000)
      read (*,  * ) k, eps, al
      write(*,1000) k, eps, al
      call fqlag(al,k,eps,t,a)
      write(*,1010)                          ! Вывод заголовка и самой
      write(*,1100) (i, T(i),A(i),i=1,k )    ! таблицы узлов и весов.
      write(*,*)
      c0=0d0
      c1=1d0
      write(*,1001)
      relmax=c0
      exact =c1
      do m=0,(2*k-1)
         s=c0
         do n=1,k
           s=s+t(n)**m*a(n)
         enddo
         exact=gamma(1+m+al)
         aer=abs(s-exact); rer=aer/exact
         relmax=max(rer,relmax)
         write(*,1011) m, exact, s, aer, rer
      enddo
      write(*,'(" #  relmax=",e9.2)') relmax
 1000 format(1x,'# k= ',i3,3x,'eps= ',D23.16,3x,' al=',d23.16 )
 1001 format(' #  n',11x,'Г(al+m+1',15x,
     >'по квадратуре',9x,'абс.',6x,'отн.')
 1011 format(1x,i4,1x,e25.17,1x,e25.17,2x,e8.2,2x,e8.2)
 1010 format(' # ',14x,'Узлы',25x,'Веса')
 1100 format(1x,i6,e25.17,3x,e25.17)
      end
```

```
! ====================================================================
! Подпрограмма вычисляет с относительной погрешностью (eps) узлы
! k-точечной квадратуры по полуоси с весом  x**(al)*exp(-x).
! После работы подпрограммы первые (k) элементов вектора (t) хранят
! найденные узлы, а в первых (k) элементах вектора (a) хранятся
! соответствующие веса.
! Алгоритм представляет переложение на язык ФОРТРАН соответствующей
! Алгол-процедуры взятой из АЛГОЛ-ПРОЦЕДУРЫ, ВЫПУСК 12, СТР.6;
! А.К.Пономаренко. Изд. Ленгос. Ун-та  19ННг.
! ....................................................................
      SUBROUTINE FQLAG(AL,K,EPS,T,A)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION T(K),A(K)
      DATA C0,C1,C3/0D0,1D0,3D0/
         R=C1
       IF(AL.EQ.C0) GO TO 5; AN=K+AL; R=dgamma(1d0+AL)
       DO N=1,K
         R=R*AN/N; AN=AN-C1
       ENDDO
    5  Q=AL-C1; Y=C1+0.3D0*AL; Z=1.26D0*AL
       I=0
      DP=K
      X=(C1+AL)*(C3+0.92D0*AL)/(C1+2.4D0*DP+1.8D0*AL)
      GO TO 30
   10 X=T(I)+(15D0+6.25D0*AL)/(C1+0.9D0*AL+2.5D0*DP)
      GO TO 30
   20 J=I-1
      DJ=J
      X=T(I)+(T(I)-T(J))/Y*((C1+2.55D0*DJ)/(1.9D0*DJ)+Z*DJ/(C1+
     +   3.5D0*DJ))
   30 E2=C1
      E3=AL+C1-X
      IF(K.EQ.1) GO TO 41
      DO 40 J=2,K
         E1=E2; E2=E3; DJ=J
         DJ1=DJ+Q
   40 E3=((DJ+DJ1-X)*E2-DJ1*E1)/DJ;
   41 E1=DP*E3-(DP+AL)*E2
      E2=E3/E1
      X=X*(C1-E2)
      IF(DABS(E2)-EPS.ge.0d0) goto 30
      I=I+1; T(I)=X
      A(I)=R*X/(E1*E1)
      if ((i-1).eq.0) goto 10;  if ((i-1).lt.0) stop
      if ((i-k).lt.0) goto 20
      if ((i-k).eq.0) return ;  if ((i-k).gt.0) stop
      RETURN
      END
```

Результаты тестирования приведены для $\mathbf{k = 10}$, $\mathbf{eps = 10^{-10}}-$ и $\alpha = \mathbf{0.5}$:

```
# k=  10   eps=  0.1000000000000000D-14    al= 0.5000000000000000D+00
#              Узлы                          Веса
     1  0.2298729805186 5618E+00     0.1754708150466 5998E+00
     2  0.9244815469866 5719E+00     0.3552233888020 7177E+00
     3  0.2099410462708 7982E+01     0.2526835596756 7824E+00
     4  0.3782880873707 2905E+01     0.8635610269533 2670E-01
     5  0.6019918027701 4603E+01     0.1510977803486 0818E-01
     6  0.8880347597996 7086E+01     0.1328215628363 5606E-02
     7  0.1247483240483 6205E+02     0.5418780021170 3528E-04
     8  0.1699084729354 2554E+02     0.8737475869187 1420E-06
     9  0.2279100289494 8946E+02     0.4019699886939 8034E-08
    10  0.3080640591705 2722E+02     0.2292221530204 6036E-11

#  n          Г(al+m+1                 по квадратуре          абс.      отн.
   0   0.8862269254527 5805E+00   0.8862269254527 5783E+00  0.22E-15  0.25E-15
   1   0.1329340388179 1370E+01   0.1329340388179 1377E+01  0.67E-15  0.50E-15
   2   0.3323350970447 8426E+01   0.3323350970447 8434E+01  0.89E-15  0.27E-15
   3   0.1163172839656 7450E+02   0.1163172839656 7452E+02  0.18E-14  0.15E-15
   4   0.5234277778455 3533E+02   0.5234277778455 3533E+02  0.00E+00  0.00E+00
   5   0.2878852778150 4416E+03   0.2878852778150 4428E+03  0.11E-12  0.39E-15
   6   0.1871254305797 7896E+04   0.1871254305797 7873E+04  0.23E-11  0.12E-14
   7   0.1403440729348 3402E+05   0.1403440729348 3398E+05  0.36E-11  0.26E-15
   8   0.1192924619946 0901E+06   0.1192924619946 0892E+06  0.87E-10  0.73E-15
   9   0.1133278388948 7833E+07   0.1133278388948 7849E+07  0.16E-08  0.14E-14
  10   0.1189942308396 2219E+08   0.1189942308396 2245E+08  0.26E-07  0.22E-14
  11   0.1368433654655 6622E+09   0.1368433654655 6586E+09  0.36E-06  0.26E-14
  12   0.1710542068319 5722E+10   0.1710542068319 5734E+10  0.12E-05  0.70E-15
  13   0.2309231792231 4262E+11   0.2309231792231 4247E+11  0.15E-04  0.66E-15
  14   0.3348386098735 5640E+12   0.3348386098735 5658E+12  0.18E-03  0.55E-15
  15   0.5189998453040 1094E+13   0.5189998453040 1260E+13  0.17E-01  0.32E-14
  16   0.8563497447516 2219E+14   0.8563497447516 2062E+14  0.16E+00  0.18E-14
  17   0.1498612053315 3325E+16   0.1498612053315 3352E+16  0.28E+01  0.18E-14
  18   0.2772432298633 3632E+17   0.2772432298633 3676E+17  0.44E+02  0.16E-14
  19   0.5406242982335 0720E+18   0.5406242982335 0592E+18  0.13E+04  0.24E-14
#  relmax= 0.32E-14
```

В книге [9] приведены, частности, таблицы узлов и весов для трёх наборов значений $\alpha = \mathbf{0(1)5}$; $\quad -\mathbf{0.75(0.25)3.50}$; $\quad -\mathbf{2/3(1/3)8/3}$, с которыми при желании можно сравнить результаты, получаемые **fqlag**

5.4 Подпрограмма gauleg и её тестирование

Подпрограммы **gauleg**, **gaulag**, **gauerm** и **gaujac** используют модуль **my_prec**:

```
module my_prec; implicit none; integer, parameter :: mp=16
end module my_prec
```

с одной единственной именованной константой **mp**, которую можно задать равной **4**, **8**, **10** или **16**, что позволит, без запоминания разнообразных соответствующих опций компиляторов, вести расчёт с одинарной, удвоенной, расширенной или четверной точностью.

Все подпрограммы в качестве результата выдают — **x**, **w** (массивы, которые в своих первых **n** элементах хранят вычисленные узлы и веса), **iter** — количество уточняющих итераций, **ier** — код завершения работы подрограмм: **0**, если требуемая точность достигнута и **1**, если её не удалось достичь за максимально возможное число итераций, которое установлено равным 20.

Для тестирования **gauleg** использовалась формула

$$\int_{\mathbf{a}}^{\mathbf{b}} \mathbf{x^m dx} = \frac{\mathbf{b^{m+1} - a^{m+1}}}{\mathbf{m+1}}, \quad (\mathbf{m = 0(1)2k - 1}).$$

```
program test_gauleg; use my_prec; implicit none
real(mp) a, b, eps; real(mp), allocatable :: t(:), w(:), zn(:), zg(:)
integer k, ier, i, iter, j
read(*, '(i10)' ) k
read(*,'(e10.3)') eps, a, b
write(*,'(" #    k=",i5   )') k; write(*,'(" #  eps=",e10.3)') eps
                                 write(*,'(" #    a=",e10.3)') a
                                 write(*,'(" #    b=",e10.3)') b
allocate(t(k),w(k),zn(0:2*k-1), zg(0:2*k-1), stat=ier)
if (ier/=0) stop 1

call gauleg(a,b,eps,t,w,k,iter,ier)

write (*, '(" #",15x,"Узлы",21x,"Веса")')
write (*, 15) (i,t(i),w(i),i=1,k)
write(*,'(" # ",32x," 1-sum(w)=",e10.3,3x,"ier=",i2,3x,"iter=",i2)')&
                                     & 1-sum(w), ier,iter

zn(0:2*k-1)=(/ ((b**(j+1))/(j+1)-(a**(j+1))/(j+1),j=0,2*k-1) /)
zg(0:2*k-1)=(/ (sum(w*t**j),j=0,2*k-1) /)
write (*,16) ' #  j','По Ньютону-Лейбницу','По Гауссу','абс','отн'
write (*,17) (j,zn(j),zg(j),abs(zn(j)-zg(j)),&
                             & abs(zn(j)-zg(j))/zn(j), j=0,2*k-1)
15 format(1x,i4,2x,e23.15,2x,e23.15)
16 format(//a,4x,a,10x,a,11x,a,8x,a/)
17 format(1x,i4,1x,d23.15,1x,d23.15,2x,d10.4,1x,d10.4)
end
```

```
subroutine gauleg(x1,x2,eps,x,w,n,iter,ier); use my_prec; implicit none
real(mp), intent(in) :: x1, x2, eps
integer,intent(in)   :: n
integer,intent(out)  :: iter, ier
real(mp), dimension(n), intent(out) :: x, w
integer, parameter :: maxit=50
real(mp), parameter :: c025=0.25_mp, c05=0.5_mp, c1=1.0_mp, c2=2.0_mp
integer j, m
real(mp) xl, xm, pi
real(mp), dimension ((n+1)/2) :: p1, p2, p3, pp, z, z1
logical , dimension ((n+1)/2) :: nokey
m=(n+1)/2; xm=c05*(x2+x1); xl=c05*(x2-x1); pi=4*atan(c1)
ier=0
z=cos(pi*((/(j,j=1,m)/)-c025)/(n+c05))
nokey=.true.
iter=1
do; where (nokey); p1=1.0_mp; p2=0.0_mp; end where
  do j=1,n
    where (nokey); p3=p2; p2=p1;
                         p1=((c2*j-c1)*z*p2-(j-c1)*p3)/j
    end where
  enddo
  where (nokey)
    pp=n*(z*p1-p2)/(z*z-c1); z1=z; z=z1-p1/pp; nokey=(abs(z-z1)>eps)
  end where
  if (.not. any(nokey)) exit
  iter=iter+1
  if (iter>maxit) exit
enddo
if (iter>maxit) ier=1
x(1:  m     )=xm-xl*z
x(n:n-m+1:-1)=xm+xl*z
w(1:  m     )=c2*xl/((c1-z**2)*pp**2)
w(n:n-m+1:-1)=w(1:m)
end subroutine gauleg
```

Результаты тестирования **gauleg** при $\mathbf{k = 10}$ и $\mathbf{eps = 10^{-10}}$:

```
#    k=   10
#  eps= 0.100E-12
#    a= 0.000E+00
#    b= 0.100E+01
#              Узлы                    Веса
   1    0.130467357414141E-01    0.333356721543417E-01
   2    0.674683166555077E-01    0.747256745752903E-01
   3    0.160295215850488E+00    0.109543181257991E+00
   4    0.283302302935376E+00    0.134633359654996E+00
   5    0.425562830509184E+00    0.147762112357376E+00
   6    0.574437169490816E+00    0.147762112357376E+00
   7    0.716697697064624E+00    0.134633359654996E+00
   8    0.839704784149512E+00    0.109543181257991E+00
   9    0.932531683344492E+00    0.747256745752903E-01
  10    0.986953264258586E+00    0.333356721543417E-01
#                                 1-sum(w)= 0.925E-14   ier= 0   iter= 4


#  j    По Ньютону-Лейбницу          По Гауссу           абс        отн

   0   0.100000000000000D+01   0.999999999999991D+00  0.9249D-14 0.9249D-14
   1   0.500000000000000D+00   0.499999999999995D+00  0.4624D-14 0.9249D-14
   2   0.333333333333333D+00   0.333333333333330D+00  0.3635D-14 0.1091D-13
   3   0.250000000000000D+00   0.249999999999997D+00  0.3141D-14 0.1256D-13
   4   0.200000000000000D+00   0.199999999999997D+00  0.2836D-14 0.1418D-13
   5   0.166666666666667D+00   0.166666666666664D+00  0.2625D-14 0.1575D-13
   6   0.142857142857143D+00   0.142857142857140D+00  0.2471D-14 0.1730D-13
   7   0.125000000000000D+00   0.124999999999998D+00  0.2354D-14 0.1883D-13
   8   0.111111111111111D+00   0.111111111111109D+00  0.2263D-14 0.2037D-13
   9   0.100000000000000D+00   0.999999999999978D-01  0.2191D-14 0.2191D-13
  10   0.909090909090909D-01   0.909090909090888D-01  0.2131D-14 0.2344D-13
  11   0.833333333333333D-01   0.833333333333313D-01  0.2081D-14 0.2498D-13
  12   0.769230769230769D-01   0.769230769230749D-01  0.2038D-14 0.2650D-13
  13   0.714285714285714D-01   0.714285714285694D-01  0.2000D-14 0.2801D-13
  14   0.666666666666667D-01   0.666666666666647D-01  0.1966D-14 0.2949D-13
  15   0.625000000000000D-01   0.624999999999981D-01  0.1935D-14 0.3096D-13
  16   0.588235294117647D-01   0.588235294117628D-01  0.1905D-14 0.3239D-13
  17   0.555555555555556D-01   0.555555555555537D-01  0.1877D-14 0.3379D-13
  18   0.526315789473684D-01   0.526315789473666D-01  0.1851D-14 0.3516D-13
  19   0.500000000000000D-01   0.499999999999982D-01  0.1825D-14 0.3650D-13
```

5.5 Подпрограмма gaulag и её тестирование

Для тестирования **gaulag** использовалась формула

$$\int_0^\infty \mathbf{x^\alpha e^{-x}\cdot x^m dx = \Gamma(1+m+\alpha)},\quad \mathbf{(m=0(1)2k-1)}.$$

```
! ========================================================================
! Программа посредством численного интегрирования функций
! f(x)=x**m (m=0 (1) 2k-1) k-точечной квадратурой Гаусса-Лаггера вида
!  [0,...) x^al exp(-x) f(x) dx = [n=1,k] A(n) f(x(n)) = Г(al+m+1),
! где Г(x) - Гамма-функция, оценивает точность расчёта узлов и весов
! квадратуры. Если они верны, то квадратурная сумма с точностью до
! погрешности их расчёта должна совпасть со значением Г(al+m+1).
! ........................................................................
      program test_gaulag; use my_prec; implicit none
      real(mp), allocatable :: x(:), w(:)
      real(mp), parameter   :: c0=0.0_mp, c1=1.0_mp
      integer i, m, k, ier, iter
      real(mp) eps, al, s, aer, rer, exact, relmax
      read (*,  * ) k, eps, al
      write(*,1000) k, eps, al
      allocate(x(k), w(k), stat=ier); if (ier/=0) stop 1
      call gaulag(x, w,al,eps,k,iter,ier)
      write(*,1010)                            ! Вывод заголовка и самой
      write(*,1100) (i, x(i),w(i),i=1,k )      ! таблицы узлов и весов.
      write(*,*)
      write(*,1001)
      relmax=c0
      exact =c1
      do m=0,(2*k-1)
         s=c0
         do i=1,k
           s=s+x(i)**m*w(i)
         enddo
         exact=gamma(1+m+al)
         aer=abs(s-exact); rer=aer/exact
         relmax=max(rer,relmax)
         write(*,1011) m, exact, s, aer, rer
      enddo
      write(*,'(" #  relmax=",e9.2)') relmax
 1000 format(1x,'# k= ',i3,3x,'eps= ',D23.16,3x,' al=',d23.16 )
 1001 format(' #  n',11x,'Г(al+m+1',15x,&
     &       'по квадратуре',9x,'абс.',6x,'отн.')
 1011 format(1x,i4,1x,e25.17,1x,e25.17,2x,e8.2,2x,e8.2)
 1010 format(' # ',14x,'Узлы',25x,'Веса')
 1100 format(1x,i6,e25.17,3x,e25.17)
      end
```

```
! ===================================================================
! gaulag(x,w,al,eps,n,iter,ier) для заданного количества узлов (n) и
! требуемой относительной погрешности их расчёта (eps) вычисляет
! узлы веса n-точечной обобщённой квадратуры Гаусса-Лагерра, помещая
! их в первые (n) элементов векторов (x) и (w) соответственно.
! Узлы располагаются в порядке уменьшения.
! ...................................................................
subroutine gaulag(x,w,al,eps,n,iter,ier); use my_prec; implicit none
integer, parameter   :: maxit=20
real(mp), intent(in) :: al, eps; integer, intent(in) :: n
real(mp), dimension(n), intent(out) :: x,w
integer, intent(out) :: iter, ier;  integer j
real(mp) anu, pi, gamln
real(mp), parameter :: C1=9.084064e-01_mp,C2=5.214976e-02_mp,&
&C3=2.579930e-03_mp,C4=3.986126e-03_mp
real(mp), dimension(n) :: rhs,r2,r3,theta
real(mp), dimension(n) :: p1,p2,p3,pp,z,z1
logical nokey(n)
pi=4.0_mp*atan(1.0_mp)
anu=4.0_mp*n+2.0_mp*al+2.0_mp
rhs=(/ (4*j-1,j=1,n) /)*pi/anu
r3=rhs**(1.0_mp/3.0_mp)
r2=r3**2
theta=r3*(C1+r2*(C2+r2*(C3+r2*C4)))
z=anu*cos(theta)**2
nokey=.true.
do iter=1,maxit
  where(nokey); p1=1.0_mp; p2=0.0_mp; end where
  do j=1,n
    where (nokey)
    p3=p2
    p2=p1
    p1=((2.0_mp*j-1.0_mp+al-z)*p2-(j-1.0_mp+al)*p3)/j
  end where
end do
where (nokey)
  pp=(n*p1-(n+al)*p2)/z; z1=z
  z=z1-p1/pp
  nokey=(abs(z-z1) > eps*z)
end where
if (.not. any(nokey)) exit
end do
ier=0; if (iter == maxit+1) ier=1
x=z
!w=-gamma(al+n)/gamma(real(n,mp))/(pp*n*p2)
w=-exp(gamln(al+n)-gamln(real(n,mp)))/(pp*n*p2)
end subroutine gaulag
```

Результаты тестирования **gaulag** при $\mathbf{k = 10}$ и $\mathbf{eps = 10^{-10}}$:

```
# k=  10   eps=  0.1000000000000000D-14    al= 0.5000000000000000D+00
#              Узлы                          Веса
     1  0.3080640591705272 3E+02     0.2292221530204709 1E-11
     2  0.2279100289494894 6E+02     0.4019699886939792 5E-08
     3  0.1699084729354255 4E+02     0.8737475869187144 6E-06
     4  0.1247483240483620 5E+02     0.5418780021170343 9E-04
     5  0.8880347597996708 6E+01     0.1328215628363564 2E-02
     6  0.6019918027701460 9E+01     0.1510977803486081 2E-01
     7  0.3782880873707290 2E+01     0.8635610269533262 7E-01
     8  0.2099410462708798 2E+01     0.2526835596756779 7E+00
     9  0.9244815469866573 6E+00     0.3552233888020720 5E+00
    10  0.2298729805186562 2E+00     0.1754708150466601 0E+00

#  n          Г(al+m+1              по квадратуре          абс.      отн.
   0   0.8862269254527580 1E+00   0.8862269254527578 5E+00  0.16E-15  0.18E-15
   1   0.1329340388179137 0E+01   0.1329340388179137 0E+01  0.30E-16  0.22E-16
   2   0.3323350970447842 6E+01   0.3323350970447842 6E+01  0.88E-17  0.26E-17
   3   0.1163172839656744 9E+02   0.1163172839656744 9E+02  0.34E-16  0.30E-17
   4   0.5234277778455352 0E+02   0.5234277778455352 0E+02  0.70E-16  0.13E-17
   5   0.2878852778150443 6E+03   0.2878852778150443 6E+03  0.62E-16  0.22E-18
   6   0.1871254305797788 3E+04   0.1871254305797788 3E+04  0.12E-14  0.62E-18
   7   0.1403440729348341 3E+05   0.1403440729348341 3E+05  0.24E-13  0.17E-17
   8   0.1192924619946090 1E+06   0.1192924619946090 1E+06  0.43E-12  0.36E-17
   9   0.1133278388948785 6E+07   0.1133278388948785 6E+07  0.80E-11  0.71E-17
  10   0.1189942308396224 8E+08   0.1189942308396224 8E+08  0.16E-09  0.13E-16
  11   0.1368433654655658 6E+09   0.1368433654655658 5E+09  0.33E-08  0.24E-16
  12   0.1710542068319573 2E+10   0.1710542068319573 1E+10  0.69E-07  0.40E-16
  13   0.2309231792231423 8E+11   0.2309231792231423 7E+11  0.15E-05  0.65E-16
  14   0.3348386098735564 6E+12   0.3348386098735564 2E+12  0.33E-04  0.98E-16
  15   0.5189998453040125 1E+13   0.5189998453040124 4E+13  0.73E-03  0.14E-15
  16   0.8563497447516206 4E+14   0.8563497447516204 8E+14  0.16E-01  0.19E-15
  17   0.1498612053315336 1E+16   0.1498612053315335 8E+16  0.36E+00  0.24E-15
  18   0.2772432298633371 8E+17   0.2772432298633371 0E+17  0.82E+01  0.30E-15
  19   0.5406242982335075 0E+18   0.5406242982335073 2E+18  0.19E+03  0.34E-15
#  relmax= 0.34E-15
```

5.6 Подпрограмма gauerm и её тестирование

Для тестирования **gauerm** использовалась формула

$$\int_{-\infty}^{\infty} \mathbf{e^{-x^2}} \cdot \mathbf{x^{2m}} \mathbf{dx} = \sqrt{\pi} \frac{\mathbf{(2m-1)!!}}{\mathbf{2^m}}, \quad \mathbf{(m = 0(1)k - 1)}.$$

```
program test_gauerm; use my_prec ! mp: константа разновидности типа real
implicit none
real(mp) eps, re, rs, a, aer, rer, pi, sp
real(mp), allocatable :: t(:), w(:)
integer n, ier, i, iter
read(*, '(i10)' ) n; read(*,'(e10.3)') eps
write(*,'(" #    n=",i5   )') n
write(*,'(" #  eps=",e10.3)') eps
allocate(t(n),w(n),stat=ier); write(*,'(" # ier=",i3)') ier;
                                    if (ier/=0) stop 1
call gauerm(t, w, eps, n, iter, ier)

write (*, '(" #",14x,"Points                    Weights")')
write (*, 15) (i,t(i),w(i),i=1,n)
write (*,'(" # iter=",i3," ier=",i3)') iter, ier
pi=4.0_mp * atan(1.0_mp)
sp=sqrt(pi)
write(*,'(" #  eps=",e10.2)') eps
write(*,'(" # ",4x,"aer(",i2,")", 4x,"rer(",i2,")")') mp, mp
a=1.0_mp
do i=0,n-1
  re=sp*a;      rs=sum(w*t**(2*i))
  a=a*(2*i+1)/2
  aer=abs(rs-re);  rer=aer/re
  write(*,'(i3,2e11.2e3)') i,aer, rer
enddo
15 format(1x,i4,2x,e23.15,2x,e23.15)
end
```

```
! ===================================================================
! gauerm(x,w,eps,n,iter,ier) для заданного количества узлов (n) и
! требуемой относительной погрешности их расчёта (eps) вычисляет
! узлы веса n-точечной квадратуры Эрмита, помещая их в первые (n)
! элементов векторов (x) и (w) соответственно. Узлы располагаются в
! порядке уменьшения.
! ...................................................................
subroutine gauerm(x,w,eps,n,iter,ier); use my_prec; implicit none
integer, parameter   :: maxit=20
integer,  intent(in) :: n
real(mp), intent(in) :: eps; real(mp), intent(out) :: x(n), w(n)
integer, intent(out) :: iter, ier
integer j,m
real(mp) :: anu, pi, pim4
real(mp), parameter :: C1=9.084064e-01_mp,C2=5.214976e-02_mp,&
&C3=2.579930e-03_mp,C4=3.986126e-03_mp
real(mp), dimension((n+1)/2) :: rhs,r2,r3,theta
real(mp), dimension((n+1)/2) :: p1,p2,p3,pp,z,z1
logical nokey((n+1)/2)
pi=4*atan(1.0_mp)
pim4=1.0_mp/pi**0.25_mp; m=(n+1)/2
 anu=2.0_mp*n+1.0_mp
 rhs=(/ (4*j-1,j=1,m) /)*pi/anu
 r3=rhs**(1.0_mp/3.0_mp)
 r2=r3**2
theta=r3*(C1+r2*(C2+r2*(C3+r2*C4)))
z=sqrt(anu)*cos(theta) ! Начальное приближение корней.
nokey=.true.           ! Корни пока ещё не уточнены окончательно.
do iter=1,maxit        ! Уточняем их по Ньютону одновременно.
  where (nokey); p1=pim4; p2=0.0; end where
  do j=1,n               ! Цикл расчёта значений полиномов Эрмита по
    where (nokey)        ! рекуррентному соотношению.
      p3=p2
      p2=p1
      p1=z*sqrt(2.0_mp/j)*p2-sqrt(real(j-1,mp)/real(j,mp))*p3
    end where
  end do
  where (nokey)
    pp=sqrt(2.0_mp*n)*p2; z1=z; z=z1-p1/pp
    nokey=(abs(z-z1) > eps)
  end where
  if (.not. any(nokey)) exit
end do
ier=0; if (iter == maxit+1) ier=1
x(1:m)=z;            x(n:n-m+1:-1)=-z
w(1:m)=2.0_mp/pp**2; w(n:n-m+1:-1)=w(1:m)
end subroutine gauerm
```

Результаты тестирования **gaulag** при $\mathbf{k = 10}$ и $\mathbf{eps = 10^{-10}}$:

```
#    n=   10
#  eps= 0.100E-32
# ier=  0
#              Points                  Weights
   1    0.343615911883774E+01    0.764043285523262E-05
   2    0.253273167423279E+01    0.134364574678123E-02
   3    0.175668364929988E+01    0.338743944554811E-01
   4    0.103661082978951E+01    0.240138611082315E+00
   5    0.342901327223705E+00    0.610862633735326E+00
   6   -0.342901327223705E+00    0.610862633735326E+00
   7   -0.103661082978951E+01    0.240138611082315E+00
   8   -0.175668364929988E+01    0.338743944554811E-01
   9   -0.253273167423279E+01    0.134364574678123E-02
  10   -0.343615911883774E+01    0.764043285523262E-05
# iter=  6 ier=  0
#  eps=  0.10E-32
#     aer(16)    rer(16)
 0  0.00E+000  0.00E+000
 1  0.96E-034  0.11E-033
 2  0.39E-033  0.29E-033
 3  0.23E-032  0.70E-033
 4  0.17E-031  0.15E-032
 5  0.99E-031  0.19E-032
 6  0.59E-030  0.21E-032
 7  0.34E-029  0.18E-032
 8  0.14E-028  0.10E-032
 9  0.00E+000  0.00E+000
```

5.7 Подпрограмма gaujac и её тестирование

Для тестирования **gaujac** использовалась формула

$$\int_{-1}^{1}(1-x)^{\alpha}\cdot(1+x)^{\beta}\cdot(P_n^{(\alpha,\beta)})^2dx=\frac{2^{\alpha+\beta+1}}{(\alpha+\beta+1+2n)}\frac{\Gamma(\alpha+n+1)\Gamma(\beta+n+1)}{\Gamma(\alpha+\beta+n+1)\Gamma(n+1)}.$$

```
program test_gaujac; use my_prec ! mp константа разновидности типа real
implicit none
real(mp) eps, al, be
real(mp), allocatable :: t(:), w(:)
integer n, ier, i, iter
read(*, '(i10)' ) n; read(*,'(e10.3)') eps, al, be
write(*,'(" #    k=",i5   )') n
write(*,'(" #  eps=",e10.3)') eps
write(*,'(" #   al=",e10.3)') al
write(*,'(" #   be=",e10.3)') be
allocate(t(n),w(n),stat=ier)
if (ier/=0) stop 1

call gaujac(t, w, al, be, eps, n, iter, ier)
write (*, '("  #",14x,"Points                    Weights")')
write (*, 15) (i,t(i),w(i),i=1,n)
write (*,'(" # iter=",i3," ier=",i3)') iter, ier

call inspect(al,be,t,w,n)
15 format(1x,i4,2x,e23.15,2x,e23.15)
end
```

```
! ================================================================
! Подпрограмма gaujac по заданным параметрам al и be полиномов Якоби,
! вычисляет узлы и веса n-точечной квадратуры Гаусса-Якоби на (-1,1).
! которые помещает в первые n элементов векторов x(n) и w(n) соответ-
! ственно (узлы упорядочиваются по убыванию).
! eps - требуемая отн.погр. расчёта узлов.
! ier - код причины завершения работы gaujac:
!  0 - eps достигнута (в iter число потребовавшихся уточнений);
!  1 - НЕТ, превышено maxit (допустимое число уточнений).
! gaujac использует модуль my_prec, импортируя из него константу mp,
! определяющей разновидность типа real.
! Использует модуль my_prec (импорт константы разновидности типа real
! и  функцию  gamln (расчёт ln(Γ(x)).
! ................................................................
subroutine gaujac(x, w, al, be, eps, n, iter, ier); use my_prec
implicit none
integer, intent(in)                 :: n
real(mp), dimension(n), intent(out) :: x, w
real(mp),intent( in)                :: al, be, eps
integer, intent(out)                :: iter, ier
integer, parameter     :: maxit=10; integer :: j
real(mp), parameter    :: c025=0.25_mp, c05=0.5_mp, c1=1.0_mp, c2=2.0_mp
real(mp)               :: pi, albe, a, c, v, gamln
real(mp), dimension(n) :: b, p1, p2, p3, pp, z, z1
logical,  dimension(n) :: notok
pi=4*atan(c1); ier=0; albe=al+be
z=cos(pi*((/(j,j=1,n)/)-c025+c05*al)/(n+c05*(albe+c1))); notok=.true.
do iter=1,maxit       ! Метод Ньютона проводится одновременно для всех
  v=c2+albe           !                                              корней
  where (notok); p1=(al-be+v*z)/c2; p2=1.0; endwhere
  do j=2,n
    a=2*j*(j+albe)*v; v=v+c2; c=c2*(j-c1+al)*(j-c1+be)*v
    where (notok)
      p3=p2; p2=p1; b=(v-c1)*(al*al-be*be+v*(v-c2)*z); p1=(b*p2-c*p3)/a
    endwhere
  enddo
  where (notok)
    pp=(n*(al-be-v*z)*p1+c2*(n+al)*(n+be)*p2)/(v*(c1-z*z))
    z1=z
    z=z1-p1/pp; notok=(abs(z-z1) > eps)            ! Формула Ньютона
  endwhere
  if (.not. any(notok)) exit
enddo
if (iter == maxit+1) ier=1
x=z
w=exp(gamln(al+n)+gamln(be+n)-gamln(n+c1)-gamln(n+al+be+c1))&
                                        &*v*c2**albe/(pp*p2)
endsubroutine gaujac
```

Результаты тестирования **gaujac** при $\mathbf{k = 10}$ и $\mathbf{eps = 10^{-10}}$:

```
#    k=   10
#  eps= 0.100E-29
#   al= 0.000E+00
#   be= 0.000E+00
 #              Points                  Weights
   1    0.973906528517172E+00    0.666713443086881E-01
   2    0.865063366688985E+00    0.149451349150581E+00
   3    0.679409568299024E+00    0.219086362515982E+00
   4    0.433395394129247E+00    0.269266719309996E+00
   5    0.148874338981631E+00    0.295524224714753E+00
   6   -0.148874338981631E+00    0.295524224714753E+00
   7   -0.433395394129247E+00    0.269266719309996E+00
   8   -0.679409568299024E+00    0.219086362515982E+00
   9   -0.865063366688985E+00    0.149451349150581E+00
  10   -0.973906528517172E+00    0.666713443086881E-01
# iter=  6 ier=  0
#      aer(16)     rer(16)
  0   0.386E-24   0.193E-24
  1   0.129E-24   0.193E-24
  2   0.771E-25   0.193E-24
  3   0.551E-25   0.193E-24
  4   0.429E-25   0.193E-24
  5   0.351E-25   0.193E-24
  6   0.297E-25   0.193E-24
  7   0.257E-25   0.193E-24
  8   0.227E-25   0.193E-24
  9   0.203E-25   0.193E-24
```

Входные аргументы **gaujac**: α, β, **eps** — требуемая относительная погрешность узлов квадратуры, **n** — число узлов.

Выходные — **x**, **w** (массивы, которые в своих первых **n** элементах хранят вычисленные узлы и веса), **iter** — количество уточняющих итераций, **ier** — код завершения работы подрограммы (**0**, если требуемая точность достигнута и **1**, если её не удалось достичь за максимально возможное число итераций, которое установлено равным 20.

5.8 Ещё раз об интерфейсе

Выше отмечалось, что при описании подпрограмм расчёта квадратур из [7] в список формальных аргументов, которые **перенимают форму фактического**, входят лишь имена векторов узлов и весов. Как следует вызывать процедуры, обладающие таким свойством?

Упростим ситуацию для уяснения. Допустим имеется функция с одним единственным формальным аргументом, который описан как массив, **перенимающий форму фактического аргумента**, т,е., например, так:

```
function odd0(x); implicit none; integer, intent(in)  :: x(:)
integer odd0
write(*,'(9i4)') x; write(*,*) 'odd0: shape(x)=',shape(x)
                    write(*,*) 'odd0:  size(x)=', size(x)
odd0=sum(x(1:size(x):2))
end function odd0
```

а вызывающая её программа такова:

```
program test0; implicit none; integer, parameter :: n=9
integer odd0, i
integer x(n) /1,2,3,4,5,6,7,8,9/
write(*,'(a$)') 'x:'
write(*,'(9i4)') x; write(*,*) 'main: shape(x)=',shape(x)
                    write(*,*) 'main:  size(x)=', size(x)
x=(/ (i,i=1,n) /)
write(*,'("odd0(x)=",i2)') odd0(x)
end
```

Нам пока неважно какую работу выполняет функция (в данном контексте она находит сумму элементов вектора с нечётными индексами). Программа **test0**, хоть и содержит элементы современного ФОРТРАНа, оформляет вызов **odd0** точно так, как делалось в старом ФОРТРАНе. Напомним, что

- функция **shape**: возвращает в качестве результата форму массива-аргумента (вектор с числом элементов равным размерности аргумента, содержимое которых равно количеству элементов по соответствующему измерению);
- функция **size**: возвращает количество элементов массива

Результат работы **test0**:

```
x:   1   2   3   4   5   6   7   8   9
 main: shape(x)=           9
 main:  size(x)=           9
odd0(x)=
 odd0: shape(x)=           0
 odd0:  size(x)=           0
 0
```

При обращении к **shape** и **size** из главной программы обе возвращают ожидаемый результат — число **9**. Однако, результат **odd0** — явно неверен (сумма элементов с нечётными индексами оказалась равна **0** вместо **25=1+3+5+7+9**. Более того, результат вызова **shape** и **size** из **odd0** тоже неверен. Причина — главная программа *не знает*, что формальный аргумент **odd0** должен перенять форму фактического аргумента и, поэтому, организует лишь просто передачу адреса, обманывая и **odd0**, и саму себя.

Дополним главную программу описанием интерфейса:

```
program test1; implicit none
interface
function odd0(x); implicit none; integer, intent(in)  :: x(:)
                                          integer odd0
end function odd0
end interface
integer, parameter :: n=9
integer i                          ! Тут уже не нужно описывать тип odd0!
integer x(n) /1,2,3,4,5,6,7,8,9/
write(*,'(a$)') 'x:'
write(*,'(9i4)') x; write(*,*) 'main: shape(x)=',shape(x)
                    write(*,*) 'main:  size(x)=', size(x)
x=(/ (i,i=1,n) /)
write(*,'("odd0(x)=",i2)') odd0(x)
end
```

```
x:   1   2   3   4   5   6   7   8   9
 main: shape(x)=           9
 main:  size(x)=           9
odd0(x)=   1   2   3   4   5   6   7   8   9
 odd0: shape(x)=           9
 odd0:  size(x)=           9
25
```

- **Вывод 1. Если формальный аргумент процедуры перенимает форму фактического, то в вызывающей программной единице обязательно указываем интерфейс такой процедуры**.
- **Вывод 2. Если пишем на современном ФОРТРАНе, то выгодно (в том или ином виде) указывать интерфейс всегда.**
- Указание интерфейса на ФОРТРАНе — то же, что и описание прототипа функций в СИ.
- Явное указание интерфейса возможно либо посредством оператора **interface**, либо посредством оператора **use**, обеспечивающего через подключение **модуля**, в котором описана процедура, действие **модульного** интерфейса.
- Пример указания модульного интерфейса для рассматриваемой выше задачи:

```
program test2; use my_mod; implicit none
integer, parameter :: n=9
integer i                          ! Тут уже не нужно описывать тип odd0!
integer x(n) /1,2,3,4,5,6,7,8,9/
write(*,'(a$)') 'x:'
write(*,'(9i4)') x; write(*,*) 'main: shape(x)=',shape(x)
                    write(*,*) 'main:  size(x)=', size(x)
x=(/ (i,i=1,n) /)
write(*,'("odd0(x)=",i2)') odd0(x)
end



module my_mod
implicit none
contains
function odd0(x); implicit none; integer, intent(in)  :: x(:)
integer odd0
write(*,'(9i4)') x; write(*,*) 'odd0: shape(x)=',shape(x)
                    write(*,*) 'odd0:  size(x)=', size(x)
odd0=sum(x(1:size(x):2))
end function odd0
end module my_mod

x:   1   2   3   4   5   6   7   8   9
 main: shape(x)=           9
 main:  size(x)=           9
odd0(x)=   1   2   3   4   5   6   7   8   9
 odd0: shape(x)=           9
 odd0:  size(x)=           9
25
```

На первый взгляд **модульный интерфейс** кажется более предпочтительным, поскольку не требует достаточно громоздких описаний заголовков процедур, необходимых при использовании оператора **interface**, и, по той же причине, форма указания для каких-то модульных процедур родового имени, позволяющего реализовать механизм перегрузки через оператор **module procedure**, несравнимо короче немодульного описания родовых имён.

5.9 Ещё раз об операторе where

В процедурах **gauleg**, **gaulag**, **gauerm** и **gaujac** встречается оператор **where**. Полезно на простом примере сравнить время его работы со временем работы соответствующего фрагмента программы, использующей оператор цикла **do** с параметром и оператором **if** внутри тела цикла. Рассмотрим задачу о изменении знака отрицательных элементов массива на противоположный:

```
program test3; implicit none
integer, parameter :: kmax=100000
integer x(100000), i, j
real t0, t1, t2
x=(/ (-i,i=1,100000) /)
write(*,*) '0:  x(1:3)=',x(1:3)
call cpu_time(t0)
do i=1, kmax
  where (x<0)  x=-x
enddo
call cpu_time(t1)
write(*,*) '1:  x(1:3)=',x(1:3)
write(*,*) t1-t0
do i=1, kmax
  do j=1,100000
    if (x(j)>0)  x(j)=-x(j)
  enddo
enddo
call cpu_time(t2)
write(*,*) '2:  x(1:3)=',x(1:3)
write(*,*) t2-t1
end
```

```
---------------------------------
        |    Уровни оптимизации
Оператор|  -O0   -O1   -O2   -O3
---------------------------------
  where |  16.5  16.6  10.0  16.6
  do if |  38.5  16.6  13.4  16.6
---------------------------------
```

6 QUANC8 – адаптивная процедура расчета интеграла.

Адаптивный — **подстраивающийся** под поведение подинтегральной функции. Подстраивание наиболее оптимально, если есть возможность расчёта значения функции в любой точке промежутка интегрирования. Поэтому в качестве одного из аргументов процедуры интегрирования должно быть имя функции, вычисляющей подинтегральное выражение.

Простейший пример адаптивного подхода к расчёту интеграла — метод двойного пересчёта (рассмотренный в пункте **3**). Уменьшение шага дробления позволяет надеяться на лучшее приближение к поведению подинтегральной функции. Недостаток метода двойного пересчёта — уменьшение шага дробления по всему промежутку интегрирования, что приводит к непроизводительным временным затратам.

Гораздо эффективнее дробить шаг на тех подучастках промежутка интегрирования, где функция очень быстро меняется и нельзя пренебречь вкладом этих подучастков в величину интеграла). Процедура **quanc8** (см. [12]) в значительной мере свободна от указанного недостатка.

6.1 Тестирующая программа

Тестирующая программа вызывает подпрограмму **quanc8** пятьдесят раз, используя поочерёдно в качестве подинтегральной функции одно из пятидесяти запрограммированных в функции **test50** выражений, соответствующее номеру **N**. Поскольку **quanc8** ориентирована на вызов подинтегральной функции лишь с одним формальным аргументом (независимой переменной интегрирования), то передача **N** (номера подинтегрального выражения) происходит не через список формальных параметров, а импортируется в **test50** через модуль **test**, в котором описана и сама **test50**.

```
      program main; use test; implicit none
      real(8) rer, aer, a, b, er, flag, res;   integer m, num
      pi=4*atan(1.0d0)
      read (*,'(e10.4)') rer, aer; write(*, 2010) rer, aer
      do m=1,50
         N=m; a=0; b=1; if (((N-29)*(N-30)*(N-31))==0) b=2*pi
                        if (((N-32)*(N-33))==0) a=-1
                        if (N==34) b=100*pi
         call quanc8(test50,a,b,aer,rer,res,er,num,flag)
         write(*,1000) N,res,er,num,flag
      enddo
      stop 0
 1000 format(1x,i3,2x,d22.15,2x,d10.4,2x,i4,2x,d10.4)
 2010 format(1x,'rer=',d25.17,4x,'aer=',d25.17//
     >       3x,'N',11x,'res',16x,'err',6x,'num',4x,'flag'/)
      end program main
```

Пределы интегрирования **a=0** и **b=1** (в большинстве случаев). Исключение составляют функции с номерами

- **29**, **30**, **31**, для которых **b** = **2**π при **a**=**0**;
- **32**, **33** с **a**=**-1** и **b**=**1**;
- **34** — **a**=**0** и **b** = **100**π

В качестве результата главная программа выдаёт таблицу, каждая строка которой содержит:

- **N** — номер функции;
- **res** — значение интеграла, накйденное **quanc8**;
- **er** — оценку погрешности его расчёта, выполненнную самой **quanc8** (т.е. на вход **quanc8** должна быть подана относительная или абсолютная погрешность расчёта интеграла, а **er** это — погрешность, которой **quanc8** смогла по *её мнению* добиться на самом деле)
- **num** — количество значений подинтегральной функции, которое потребовалось вычислить **quanc8** прежде чем она завершила свою работу (либо достигла желаемой точности, либо *убедилась*, что не в состоянии её достичь);
- **flag** — индикатор надёжности результата.

Напомним, что старых версиях ФОРТРАНа не было программной единицы компиляции **module**. Так что передачу параметра **N** приходилось осуществлять через **common**-блок (средство, нацеленное на решение той же задачи (передача дополнительных параметров процедурам), но не такое безопасное как **module**.

Тестирующие функции.

N	$\mathbf{f_N(x)}$	**a**	**b**	$\int_a^b f(x)dx$
1	1	0	1	1
2	$x-2$	0	1	-1.5
3	x^2-2x+3	0	1	2.3(3)
4	x^3-2x^2+3x-4	0	1	-2.916(6)
5	$x^4-2x^3+3x^2-4x+5$	0	1	3.7
6	$x \cdot f_5(x)-6$	0	1	-4.316(6)
7	$x \cdot f_6(x)+7$	0	1	5.07619047619048
8	$x \cdot f_7(x)-8$	0	1	-5.71071428571429
9	$x \cdot f_8(x)+9$	0	1	6.45634920634921
10	$x \cdot f_9(x)-10$	0	1	-7.10198412698413
11	$x \cdot f_{10}(x)+11$	0	1	7.83852813852814
12	$x \cdot f_{11}(x)-12$	0	1	-8.49173881673882
13	$x \cdot f_{12}(x)+13$	0	1	9.22187257187257
14	$x \cdot f_{13}(x)-14$	0	1	-9.88057775557775
15	$x \cdot f_{14}(x)+15$	0	1	10.6059496059496
16	$x \cdot f_{15}(x)-16$	0	1	-11.2688214563196
17	$x \cdot f_{16}(x)+17$	0	1	11.9905168361051
18	$x \cdot f_{17}(x)-18$	0	1	-12.6566566603315
19	$x \cdot f_{18}(x)+19$	0	1	13.3754280635086
20	$x \cdot f_{19}(x)-20$	0	1	-14.0441994666839

N	$\mathbf{f_N(x)}$	**a**	**b**	$\int_a^b f(x)dx$
21	e^x	0	1	1.71828182845905
22	$\sin \pi x$	0	1	0.636619772367584
23	$\cos x$	0	1	0.841470984807897
24	$x \neq 0: \quad \dfrac{x}{e^x - 1}$ $x = 0: \quad 1$	0	1	0.777504634112248
25	$\dfrac{1}{1 + x^2}$	0	1	0.785398163397448
26	$\dfrac{2}{2 + \sin 10\pi x}$	0	1	1.15470053837926
27	$\dfrac{1}{1 + x^4}$	0	1	0.866972987339912
28	$\dfrac{1}{1 + e^x}$	0	1	0.379885493041722
29	$x \sin 30x \cos x$	0	2π	-0.209672479661163
30	$x \sin 30x \cos 50x$	0	2π	0.117809847081320
31	$x \neq 2\pi: \dfrac{x \sin 30x}{\sqrt{1 - \dfrac{x^2}{4\pi^2}}}$ $x = 2\pi: \quad 0$	0	2π	-2.54325961832759D
32	$\dfrac{23}{25 \cosh x - \cos x}$	-1	1	0.479428226688802
33	$\dfrac{1}{x^4 + x^2 + 0.9}$	-1	1	1.58223296372755
34	$(\sin x)\sqrt{\lvert(100\pi)^2 - x^2\rvert}$	0	100π	298.427327777469

N	$\mathbf{f_N(x)}$	**a**	**b**	$\int_a^b f(x)dx$
35	$\frac{1}{1+x}$	0	1	0.693147180559946
36	$\sqrt{x}$	0	1	0.666666666666660
37	$\sqrt{\sqrt{x}}$	0	1	0.799999999999977
38	$\sqrt{\sqrt{\sqrt{x}}}$	0	1	0.888888888888288
39	$\sqrt{\sqrt{\sqrt{\sqrt{x}}}}$	0	1	0.941176470585109
40	$\sqrt{\lvert x^2 - 0.25\rvert}$	0	1	0.464742505625965
41	$x\sqrt{x}$	0	1	0.400000000000035
42	$y = \lvert x^2 - 0.25\rvert : y * \sqrt{y}$	0	1	0.148871621223236
43	$y = \sqrt{x} : x * x * y$	0	1	0.285714285714191
44	$y = \lvert x^2 - 0.25\rvert; y^2\sqrt{y}$	0	1	0.0655147683952976
45	$int(10x)$	0	1	4.50027227038675
46	$x \in [0.000, 0.333] : \quad x$ $x \in [0.333, 0.667] : 1 + x$ $x \in [0.667, 1.000] : 2 + x$	0	1	1.50000000000000
47	$x \in [0.00, 0.49] : -1000 \cdot (x^2 - x)$ $x \in [0.49, 0.50] : \quad 0$ $x \in [0.50, 1.00] : -1000 \cdot (x^2 - x)$	0	1	166.666666666667
48	$x \in [0.00, e^1 - 2] : 1/(2 + x)$ $x \in [e^1 - 2, 1.00] : \quad 0$	0	1	0.306852819429457
49	$10000 * (x - 0.10) * (x - 0.11) *$ $* (x - 0.12) * (x - 0.13)$	0 0	1 1	1085.25266666667
50	$sin(100\pi x)$	0	1	0.183602412484921D-14

6.2 Модуль расчёта подинтегральных функций

```
      module test; implicit none; integer N; real(8) pi
      contains
! ====================================================================
! Функция TECT50 (X), используемая главной программой как имя функции
! расчёта подинтегрального выражения (для тестирования quanc8) по
! аргументу X и номеру N (1<=N<=50) выражения, вычисляет его значение.
! Имена N и pi экспортируются из модуля test в главную программу
! ЛИТ-РА: COMMUNICATIONS OF THE ACM  MAY 1972 VOLUME 15 NUMBER 5 СТР.34
! --------------------------------------------------------------------
      function test50(X); real(8) test50, x, x2, y
      real(8) f(50), c(20), c30, c50, c23, c25, c09, c100, c025
      real(8) c0, c333, c667, c049, c05, ce2;  integer i, j
      data c /1D0,2D0,3D0,4D0,5D0,6D0,7D0,8D0,9D0,10D0,11D0,12D0,13D0,
     ,                              14D0,15D0,16D0,17D0,18D0,19D0,20D0/
      data c0, c333, c667, c049, c05 /0D0,0.333D0,0.667D0,0.49D0,0.5D0/
      data c09, c100, c025    /0.9D0, 100D0, 0.25D0/
      data c30, c50, c23, c25 /30D0,50D0,23D0,25D0/
      data ce2 /0.71828182845945D0/
      F(1)=c(1)
      select case (N)
        case(:20); do i=2,n; f(i)=x*f(i-1)+(1-mod(i-1,2)*2)*c(i); enddo
        case(21); f(N)=exp(x);
        case(22); f(N)=sin(pi*x);
        case(23); f(N)=cos(x);
        case(24); if(c(1)/=(c(1)+x)) then; f(N)=x/(exp(x)-c(1))
                                      else; f(N)=c(1)
                  endif
        case(25); f(N)=c(1)/(c(1)+x*x)
        case(26); f(N)=c(2)/(c(2)+sin(c(10)*pi*x))
        case(27); f(N)=c(1)/(c(1)+X**4)
        case(28); f(N)=c(1)/(c(1)+exp(x))
        case(29); f(N)=x*sin(c30*x)*cos(x)
        case(30); f(N)=X*sin(c30*x)*cos(c50*X)
        case(31); if (c(1)/=(c(1)+(c(4)*pi*pi-x*x))) then
                    f(N)=x*sin(c30*x)/sqrt(c(1)-x*x/(c(4)*pi*pi))
                    else; f(N)=c0
                  endif
        case(32); f(N)=c23/c25*cosh(X)-cos(X)
        case(33); x2=x*x; f(N)=c(1)/(x2*x2+x2+c09)
        case(34); f(N)=sin(x)*sqrt(abs((c100*pi)**2-x*x))
        case(35); f(N)=c(1)/(c(1)+x)
        case(36:39); f(36)=sqrt(x); do i=37,n; f(i)=sqrt(f(i-1)); enddo
        case(40); f(N)=sqrt(abs(x*x -c025))
        case(41); f(N)=x*sqrt(x)
        case(42); y=abs(x*x-c025); f(N)=y*sqrt(y)
        case(43); y=sqrt(x);       f(N)=x*x*y
```

```
  case(44); Y=abs(x*x-c025); f(N)=y*y*sqrt(y)
  case(45); j=int(c(10)*x);  f(N)=J
  case(46); if ((x>= c0) .and.(x<=c333)) f(N)=x
            if ((x>c333) .and.(x<=c667)) f(N)=c(1)+x
            if ((x>c667) .and.(x<=c(1))) f(N)=c(2)+x
  case(47); f(N)=c0; if((x<=c049).or.(x>=c05)) f(N)=-1000*(x*x-x)
  case(48); f(N)=c0; if( x<=ce2 ) f(N)=c(1)/(c(2)+x)
  case(49); f(N)=10000*(x-.1d0)*(x-.11d0)*(x-.12d0)*(x-.13d0)
  case(50); f(N)=sin(c100*pi*x)
end select
test50=f(N)
return
end function test50
end module test
```

6.3 Подпрограмма quanc8

```
! ====================================================================
! Подпрограмма QUANC8 - адаптивная подпрограмма, основанная
!                       на формуле НЬЮТОНА-КОТЕСА 8-ГО ПОРЯДКА.
! ВХОДНАЯ ИНФОРМАЦИЯ:
! 1. FUN - FUN(x) реализует расчёт подинтегральной функции;
! 2. A - нижний предел интегрирования;
! 3. B - верхний (B может быть меньше A);
! 4. RELERR - граница отн. погр. (неотрицательная)
! 5. ABSERR - граница абс. погр. (неотрицательная)
!
! ВЫХОДНАЯ ИНФОРМАЦИЯ:
! 6. RESULT - приближение к интегралу, удовлетворяющее (как надеемся)
!             менее жесткой из двух границ погрешностей;
! 7. ERREST - оценка фактической ошибки;
! 8. NOFUN  - число значений функции, использованных при расчёте;
! 9. FLAG   - индикатор надёжности. Если FLAG=0, то RESULT, вероятно,
!                         удовлетворяет заданной границе погрешности.
!    Если FLAG=XXX.YYY, то
!        XXX=числу интервалов, для которых сходимость не  достигнута,
!      0.YYY=части основного интервала, оставшейся для обработки в тот
!      момент, когда программа подошла к предельному для NOFUN значению.
! Подпрограмма взята из книги: ВЖ.ФОРСАЙТ, М.МАЛЬКОЛЬМ, К.МОУЛЕР
! Машинные методы математических вычислений, МИР, МОСКВА 1980 СТР. 118
! ....................................................................
      SUBROUTINE QUANC8(FUN,A,B,ABSERR,RELERR,RESULT,ERREST,NOFUN,FLAG)
      IMPLICIT REAL*8(A-H,O-Z)
      external fun
      DIMENSION QRIGHT(31),F(16),X(16),FSAVE(8,30),XSAVE(8,30)
      LEVMIN=1
      LEVMAX=30
      LEVOUT=6
      NOMAX=5000
      NOFIN=NOMAX-8*(LEVMAX-LEVOUT+2**(LEVOUT+1))
      W0= 3.956D3/1.4175D4
      W1=2.3552D4/1.4175D4
      W2=-3.712D3/1.4175D4
      W3=4.1984D4/1.4175D4
      W4=-1.816D4/1.4175D4
      FLAG=0D0
      RESULT=0.0D0
      COR11= 0.0D0
      ERREST=0.0D0
      AREA = 0.0D0
      NOFUN=0
      IF(A.EQ.B) RETURN
      LEV=0
```

```
   NIM=1
   X0 =A
   X(16)=B
   QPREV=0.0D0
   F0=FUN(X0)
   STONE=(B-A)/16.0D0
   X(8)=(X0+X(16))/2.0D0
   X(4)=(X0+X(8))/2.0D0
   X(12)=(X(8)+X(16))/2.0D0
   X(2)=(X0+X(4))/2.0D0
   X(6)=(X(4)+X(8))/2.0D0
   X(10)=(X(8)+X(12))/2.0D0
   X(14)=(X(12)+X(16))/2.0D0
   DO 25 J=2,16,2
25    F(J)=FUN(X(J))
   NOFUN=9
30 X(1)=(X0+X(2))/2.0D0
   F(1)=FUN(X(1))
   DO 35 J=3,15,2
      X(J)=(X(J-1)+X(J+1))/2.0D0
35    F(J)=FUN(X(J))
   NOFUN=NOFUN+8
   STEP=(X(16)-X0)/1.6D1
   QLEFT=(W0*(F0+F(8))+W1*(F(1)+F(7))+W2*(F(2)+F(6))+
  +W3*(F(3)+F(5))+W4*F(4))*STEP
   QRIGHT(LEV+1)=(W0*(F(8)+F(16))+W1*(F(9)+F(15))+
  +W2*(F(10)+F(14))+W3*(F(11)+F(13))+W4*F(12))*STEP
   QNOW=QLEFT+QRIGHT(LEV+1)
   QDIFF=QNOW-QPREV
   AREA=AREA+QDIFF
   ESTERR=DABS(QDIFF)/1.023D3
   TOLERR=DMAX1(ABSERR,RELERR*DABS(AREA))*
  *(STEP/STONE)
   IF(LEV.LT.LEVMIN) GO TO 50
   IF(LEV.GE.LEVMAX) GO TO 62
   IF(NOFUN.GT.NOFIN) GO TO 60
   IF(ESTERR.LE.TOLERR) GO TO 70
50 NIM=2*NIM
   LEV=LEV+1
   DO 52 I=1,8
      FSAVE(I,LEV)=F(I+8)
52    XSAVE(I,LEV)=X(I+8)
   QPREV=QLEFT
   DO 55 I=1,8
      J=-I
      F(2*J+18)=F(J+9)
55    X(2*J+18)=X(J+9)
   GO TO 30
```

```
   60 NOFIN=2*NOFIN
      LEVMAX=LEVOUT
      FLAG=FLAG+(B-X0)/(B-A)
      GO TO 70
   62 FLAG=FLAG+1D0
   70 RESULT=RESULT+QNOW
      ERREST=ERREST+ESTERR
      COR11=COR11+QDIFF/1.023D3
   72 IF(NIM.EQ.2*(NIM/2)) GO TO 75
      NIM=NIM/2
      LEV=LEV-1
      GO TO 72
   75 NIM=NIM+1
      IF(LEV.LE.0) GO TO 80
      QPREV=QRIGHT(LEV)
      X0=X(16)
      F0=F(16)
      DO 78 I=1,8
         F(2*I)=FSAVE(I,LEV)
   78    X(2*I)=XSAVE(I,LEV)
      GO TO 30
   80 RESULT=RESULT+COR11
      IF(ERREST.EQ.0.0D0) RETURN
   82 TEMP=DABS(RESULT)+ERREST
      IF(TEMP.NE.DABS(RESULT)) RETURN
      ERREST=2.0D0*ERREST
      GO TO 82
      END
```

6.4 Makefile

```
  comp   :=gfortran
  pattern:=*.f
  source :=$(wildcard $(pattern))
  object :=$(patsubst %.f, %.o, $(source))
    main : $(object)
   $(comp)  $^ -o $@
%.o %.mod : %.f
   $(comp) -c -Wall $<
    main.o : test.mod
   clear:
rm -f *.o  test.mod main
  result: main input
  ./main < input > result
```

6.5 Результаты пропуска

6.5.1 aer=rer=1.0d-03

rer= 0.10000000000000000D-02 aer= 0.10000000000000000D-02

N	res	err	num	flag
1	0.100000000000000D+01	0.0000D+00	33	0.0000D+00
2	-0.150000000000000D+01	0.0000D+00	33	0.0000D+00
3	0.233333333333333D+01	0.0000D+00	33	0.0000D+00
4	-0.291666666666667D+01	0.0000D+00	33	0.0000D+00
5	0.370000000000000D+01	0.0000D+00	33	0.0000D+00
6	-0.431666666666667D+01	0.6668D-15	33	0.0000D+00
7	0.507619047619048D+01	0.4445D-15	33	0.0000D+00
8	-0.571071428571429D+01	0.0000D+00	33	0.0000D+00
9	0.645634920634921D+01	0.0000D+00	33	0.0000D+00
10	-0.710198412698413D+01	0.0000D+00	33	0.0000D+00
11	0.783852813852814D+01	0.2039D-11	33	0.0000D+00
12	-0.849173881673882D+01	0.7138D-11	33	0.0000D+00
13	0.922187257187441D+01	0.2610D-10	33	0.0000D+00
14	-0.988057775556946D+01	0.7052D-10	33	0.0000D+00
15	0.106059496059842D+02	0.1697D-09	33	0.0000D+00
16	-0.112688214562110D+02	0.3545D-09	33	0.0000D+00
17	0.119905168364091D+02	0.6791D-09	33	0.0000D+00
18	-0.126566566596043D+02	0.1198D-08	33	0.0000D+00
19	0.133754280650892D+02	0.1987D-08	33	0.0000D+00
20	-0.140441994635363D+02	0.3124D-08	33	0.0000D+00
21	0.171828182845905D+01	0.1945D-15	33	0.0000D+00
22	0.636619772367584D+00	0.3675D-13	33	0.0000D+00
23	0.841470984807897D+00	0.6251D-16	33	0.0000D+00
24	0.777504634112248D+00	0.8335D-16	33	0.0000D+00
25	0.785398163396957D+00	0.6172D-12	33	0.0000D+00
26	0.116025264397640D+01	0.8024D-04	33	0.0000D+00
27	0.866972987345040D+00	0.4204D-11	33	0.0000D+00
28	0.379885493041722D+00	0.3115D-16	33	0.0000D+00
29	0.418877045504827D+01	0.1971D-04	33	0.0000D+00
30	-0.171799545781226D+00	0.7573D-03	65	0.0000D+00
31	0.946812242719829D+01	0.2048D-03	33	0.0000D+00
32	0.479428226688802D+00	0.2205D-14	33	0.0000D+00
33	0.158223296638773D+01	0.1426D-08	33	0.0000D+00
34	0.298400172397803D+03	0.2720D+01	481	0.0000D+00
35	0.693147180560008D+00	0.1001D-12	33	0.0000D+00
36	0.666340017525455D+00	0.5849D-06	33	0.0000D+00
37	0.798214191577534D+00	0.2409D-05	33	0.0000D+00
38	0.884911934614357D+00	0.4597D-05	33	0.0000D+00
39	0.935281407918025D+00	0.6279D-05	33	0.0000D+00
40	0.464089211462531D+00	0.1170D-05	33	0.0000D+00
41	0.400001116157714D+00	0.5104D-08	33	0.0000D+00

```
42   0.148873853942456D+00  0.1022D-07    33  0.0000D+00
43   0.285714269249554D+00  0.1677D-09    33  0.0000D+00
44   0.655147353458661D-01  0.3408D-09    33  0.0000D+00
45   0.451742563025717D+01  0.1705D-04    33  0.0000D+00
46   0.150000000000000D+01  0.1811D-03    33  0.0000D+00
47   0.166666666666667D+03  0.0000D+00    33  0.0000D+00
48   0.296148169267686D+00  0.3627D-04    33  0.0000D+00
49   0.108525266666667D+04  0.0000D+00    33  0.0000D+00
50   0.122140003764099D-14  0.1228D-17    33  0.0000D+00
```

6.6 Результаты пропуска

6.6.1 aer=rer=1.0d-06

rer= 0.99999999999999995D-06 aer= 0.99999999999999995D-06

N	res	err	num	flag
1	0.100000000000000D+01	0.0000D+00	33	0.0000D+00
2	-0.150000000000000D+01	0.0000D+00	33	0.0000D+00
3	0.233333333333333D+01	0.0000D+00	33	0.0000D+00
4	-0.291666666666667D+01	0.0000D+00	33	0.0000D+00
5	0.370000000000000D+01	0.0000D+00	33	0.0000D+00
6	-0.431666666666667D+01	0.6668D-15	33	0.0000D+00
7	0.507619047619048D+01	0.4445D-15	33	0.0000D+00
8	-0.571071428571429D+01	0.0000D+00	33	0.0000D+00
9	0.645634920634921D+01	0.0000D+00	33	0.0000D+00
10	-0.710198412698413D+01	0.0000D+00	33	0.0000D+00
11	0.783852813852814D+01	0.2039D-11	33	0.0000D+00
12	-0.849173881673882D+01	0.7138D-11	33	0.0000D+00
13	0.922187257187441D+01	0.2610D-10	33	0.0000D+00
14	-0.988057775556946D+01	0.7052D-10	33	0.0000D+00
15	0.106059496059842D+02	0.1697D-09	33	0.0000D+00
16	-0.112688214562110D+02	0.3545D-09	33	0.0000D+00
17	0.119905168364091D+02	0.6791D-09	33	0.0000D+00
18	-0.126566566596043D+02	0.1198D-08	33	0.0000D+00
19	0.133754280650892D+02	0.1987D-08	33	0.0000D+00
20	-0.140441994635363D+02	0.3124D-08	33	0.0000D+00
21	0.171828182845905D+01	0.1945D-15	33	0.0000D+00
22	0.636619772367584D+00	0.3675D-13	33	0.0000D+00
23	0.841470984807897D+00	0.6251D-16	33	0.0000D+00
24	0.777504634112248D+00	0.8335D-16	33	0.0000D+00
25	0.785398163396957D+00	0.6172D-12	33	0.0000D+00
26	0.115469899811973D+01	0.9534D-07	193	0.0000D+00
27	0.866972987345040D+00	0.4204D-11	33	0.0000D+00
28	0.379885493041722D+00	0.3115D-16	33	0.0000D+00
29	-0.209672593490016D+00	0.2456D-06	513	0.0000D+00
30	0.117809701831373D+00	0.9429D-07	1857	0.0000D+00
31	-0.660467331195094D+00	0.8370D-06	657	0.0000D+00
32	0.479428226688802D+00	0.2205D-14	33	0.0000D+00
33	0.158223296638773D+01	0.1426D-08	33	0.0000D+00
34	0.298435673385920D+03	0.2045D-03	1073	0.0000D+00
35	0.693147180560008D+00	0.1001D-12	33	0.0000D+00
36	0.666551178754977D+00	0.2068D-06	49	0.0000D+00
37	0.799999691708958D+00	0.4164D-09	193	0.0000D+00
38	0.888888888882063D+00	0.3683D-12	449	0.0000D+00
39	0.941176470584722D+00	0.2342D-12	497	0.2000D+01
40	0.464511530167038D+00	0.4136D-06	65	0.0000D+00
41	0.400001116157714D+00	0.5104D-08	33	0.0000D+00

```
42   0.148873853942456D+00  0.1022D-07    33  0.0000D+00
43   0.285714269249554D+00  0.1677D-09    33  0.0000D+00
44   0.655147353458661D-01  0.3408D-09    33  0.0000D+00
45   0.450027227038675D+01  0.2665D-06  3905  0.2710D+02
46   0.150000000000000D+01  0.3771D-08   193  0.0000D+00
47   0.166666666666667D+03  0.0000D+00    33  0.0000D+00
48   0.308870179751229D+00  0.8270D-08    65  0.0000D+00
49   0.108525266666667D+04  0.0000D+00    33  0.0000D+00
50   0.122140003764099D-14  0.1228D-17    33  0.0000D+00
```

6.7 Результаты пропуска

6.7.1 aer=rer=1.0d-12

rer= 0.10000000000000000D-12 aer= 0.10000000000000000D-12

N	res	err	num	flag
1	0.100000000000000D+01	0.0000D+00	33	0.0000D+00
2	-0.150000000000000D+01	0.0000D+00	33	0.0000D+00
3	0.233333333333333D+01	0.0000D+00	33	0.0000D+00
4	-0.291666666666667D+01	0.0000D+00	33	0.0000D+00
5	0.370000000000000D+01	0.0000D+00	33	0.0000D+00
6	-0.431666666666667D+01	0.6668D-15	33	0.0000D+00
7	0.507619047619048D+01	0.4445D-15	33	0.0000D+00
8	-0.571071428571429D+01	0.0000D+00	33	0.0000D+00
9	0.645634920634921D+01	0.0000D+00	33	0.0000D+00
10	-0.710198412698413D+01	0.0000D+00	33	0.0000D+00
11	0.783852813852814D+01	0.1991D-14	65	0.0000D+00
12	-0.849173881673882D+01	0.7594D-14	65	0.0000D+00
13	0.922187257187257D+01	0.2728D-13	65	0.0000D+00
14	-0.988057775557775D+01	0.7836D-13	65	0.0000D+00
15	0.106059496059496D+02	0.1995D-12	65	0.0000D+00
16	-0.112688214563196D+02	0.1136D-12	65	0.0000D+00
17	0.119905168361051D+02	0.9659D-13	81	0.0000D+00
18	-0.126566566603315D+02	0.6987D-12	65	0.0000D+00
19	0.133754280635086D+02	0.1840D-12	81	0.0000D+00
20	-0.140441994666839D+02	0.2347D-12	81	0.0000D+00
21	0.171828182845905D+01	0.1945D-15	33	0.0000D+00
22	0.636619772367584D+00	0.3675D-13	33	0.0000D+00
23	0.841470984807897D+00	0.6251D-16	33	0.0000D+00
24	0.777504634112248D+00	0.8335D-16	33	0.0000D+00
25	0.785398163397448D+00	0.3394D-15	65	0.0000D+00
26	0.115470053837926D+01	0.2392D-13	865	0.0000D+00
27	0.866972987339912D+00	0.1916D-13	65	0.0000D+00
28	0.379885493041722D+00	0.3115D-16	33	0.0000D+00
29	-0.209672479661163D+00	0.1301D-12	2529	0.0000D+00
30	0.117809847081319D+00	0.1992D-05	4321	0.3653D+02
31	-0.254325961832759D+01	0.1297D-11	3825	0.2000D+02
32	0.479428226688802D+00	0.2205D-14	33	0.0000D+00
33	0.158223296372755D+01	0.3944D-13	97	0.0000D+00
34	0.298427327777469D+03	0.5388D-04	3953	0.1318D+02
35	0.693147180559946D+00	0.2444D-14	49	0.0000D+00
36	0.666666666666660D+00	0.5149D-14	561	0.2000D+01
37	0.799999999999978D+00	0.1212D-14	657	0.2000D+01
38	0.888888888888288D+00	0.1630D-14	753	0.2000D+01
39	0.941176470585109D+00	0.3933D-14	833	0.2000D+01
40	0.464742505625965D+00	0.9391D-14	1057	0.4000D+01
41	0.400000000000035D+00	0.2833D-13	225	0.0000D+00

42	0.148871621223236D+00	0.1016D-13	417	0.0000D+00
43	0.285714285714191D+00	0.7044D-14	113	0.0000D+00
44	0.655147683952976D-01	0.3025D-14	193	0.0000D+00
45	0.450027227038675D+01	0.2665D-06	3905	0.2710D+02
46	0.150000000000000D+01	0.3666D-12	961	0.4000D+01
47	0.166666666666667D+03	0.0000D+00	33	0.0000D+00
48	0.306852819429457D+00	0.6218D-13	497	0.2000D+01
49	0.108525266666667D+04	0.0000D+00	33	0.0000D+00
50	0.122140003764099D-14	0.1228D-17	33	0.0000D+00

6.8 Графики подинтегральных функций.

На рисунках представлены графики пятидесяти подинтегральных функций, используемых для тестирования **quanc8**. Первый рисунок получен когда число точек, используемых для построения каждого графика, равнялось **100** (согласно установке **set samples 100** gnuplot-скрипта); она же действует и по умолчанию). Второй рисунок соответствует установке **set samples 10000**. Как видно, некоторые графики первого рисунка разительно отличаются от соответствующих им графиков второго (например, $f_{30}(x)$, $f_{34}(x)$, f_{50}).

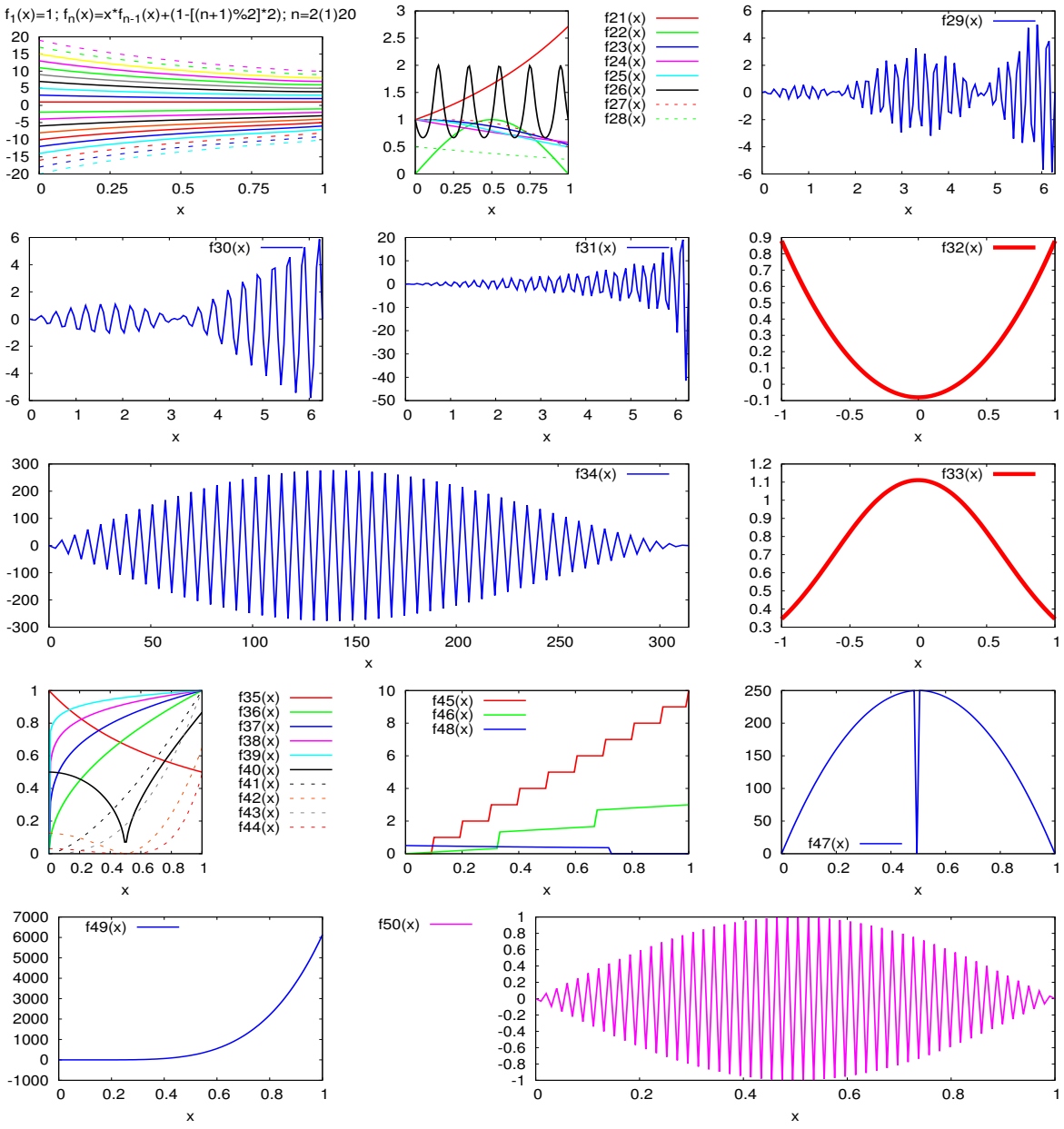


Рис. 1: Графики тестирующих функций при **set samples 100**

Причина в том, что скорость изменения этих подинтегральных функций на значительной части области их определения не может быть адекватно отражена точками дискретизации аргумента первого рисунка (просто две соседние точки дискретизации не в состоянии объективно отразить поведение функции между ними, а оно существенно принципиально). На втором рисунке точки дискретизации аргумента расположены в сто раз чаще и отражают ситуацию более адекватно.

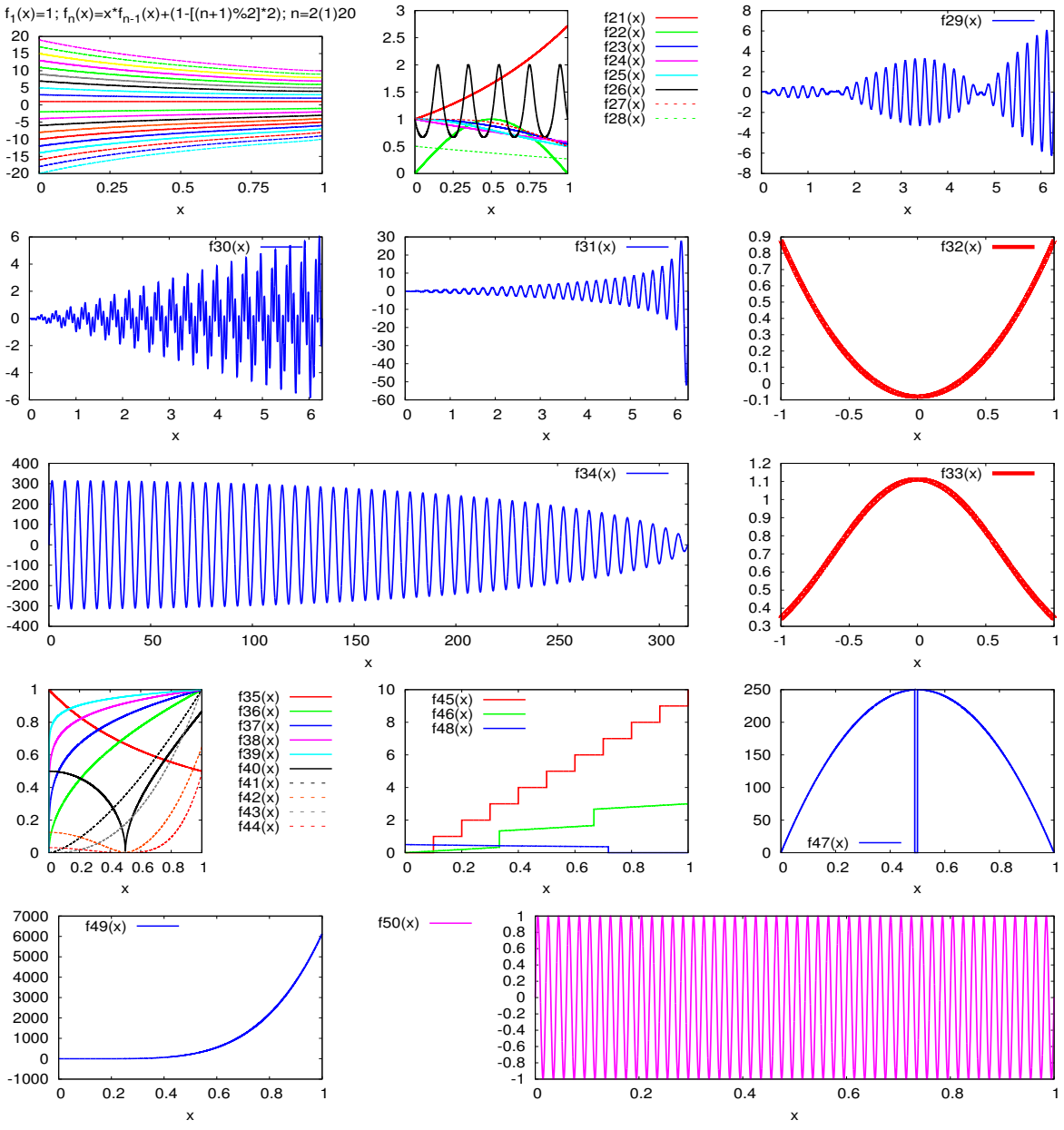


Рис. 2: Графики тестирующих функций при **set samples 10000**

Так что при оценке качества численного интегрирования полезнее ориентироваться на второй рисунок. Установки умолчания графпакетов не всегда способны адекватно воссоздать графическую иллюстрацию — иногда установки надо подкорректировать.

6.8.1 gnupot-скрипт построения рисунков 1 и 2

```
set terminal postscript eps enhance 32; set output 'gr50_10B.eps'
set samples 100; set xlabel "x"
ce2=0.71828182845945e0
f1(x)=1; f2(x)=x*f1(x)-2; f3(x)=x*f2(x)+3; f4(x)=x*f3(x)-4; f5(x)=x*f4(x)+5
f6(x)=x*f5(x)-6;          f7(x)=x*f6(x)+7; f8(x)=x*f7(x)-8; f9(x)=x*f8(x)+9
f10(x)= x*f9(x)-10; f11(x)=x*f10(x)+11; f12(x)=x*f11(x)-12; f13(x)=x*f12(x)+13
f14(x)=x*f13(x)-14; f15(x)=x*f14(x)+15; f16(x)=x*f15(x)-16; f17(x)=x*f16(x)+17
f18(x)=x*f17(x)-18; f19(x)=x*f18(x)+19; f20(x)=x*f19(x)-20;
f21(x)=exp(x);      f22(x)=sin(pi*x);   f23(x)=cos(x)
f24(x)=1.0!=1+x ? x/(exp(x)-1.0) : 1.0; f25(x)=1.0/(1.0+x*x)
f26(x)=2.0/(2.0+sin(10*pi*x)); f27(x)=1.0/(1.0+x**4); f28(x)=1.0/(1.0+exp(x))
f29(x)=x*sin(30*x)*cos(x);                  f30(x)=x*sin(30*x)*cos(50*x)
f31(x)=(1.0!=(1.0+4.0*pi*pi-x*x)) ? x*sin(30*x)/sqrt(1.0-x*x/(4.0*pi*pi)) : 0.0
f32(x)=23.0/25.0*cosh(x)-cos(x);            f33(x)=1.0/(x**4+x**2+0.9)
f34(x)=sin(x)*sqrt(abs((100.0*pi)**2-x*x)); f35(x)=1.0/(1.0+x)
f36(x)=sqrt(x); f37(x)=sqrt(f36(x)); f38(x)=sqrt(f37(x)); f39(x)=sqrt(f38(x))
f40(x)=sqrt(abs(x*x-0.25)); f41(x)=x*sqrt(x)
f42(x)=abs(x*x-0.25)*sqrt(abs(x*x-0.25)); f43(x)=x*x*sqrt(x)
f44(x)=abs(x*x-0.25)**2*sqrt(abs(x*x-0.25)); f45(x)= int(10.0*x)
f46(x)=((x>=0.0)&&(x<0.333)) ? x :\
       ((x>0.333)&&(x<0.667)) ? 1.0+x :\
       ((x>0.667) && (x<=1)) ? 2+x : 0.0
f47(x)=((x<=0.49)||(x>=0.5)) ? -1000e0*(x*x-x) : 0.0
f48(x)=(x<=ce2) ? 1.0/(2.0+x)  : 0.0
f49(x)=10000e0*(x-0.1)*(x-0.11)*(x-0.12)*(x-0.13); f50(x)=sin(100*pi*x)
set size 3,5                   # размер окна multiplot
set origin 0,0                 # начало его координат
set multiplot                  # установка режима multiplot
set size 1,1; set origin 0,4   # размер и координаты рисунка (1, 1)
set xtics 0,0.25,1; set nokey  # оцифровка X; легенда не нужна
set title "f_1(x)=1; f_n(x)=x*f_{n-1}(x)+(1-[(n+1)%2]*2); n=2(1)20"
a=0; b=1; plot [a:b] f1(x)  lt -1 lc  1 lw 3, f2(x)  lt -1 lc  2 lw 3,\
                     f3(x)  lt -1 lc  3 lw 3, f4(x)  lt -1 lc  4 lw 3,\
           f5(x)  lt -1 lc  5 lw 3, f6(x)  lt -1 lc  0 lw 3,\
           f7(x)  lt -1 lc  7 lw 3, f8(x)  lt -1 lc  8 lw 3,\
           f9(x)  lt -1 lc  9 lw 3, f10(x) lt -1 lc 10 lw 3,\
           f11(x) lt -1 lc 11 lw 3, f12(x) lt -1 lc 12 lw 3,\
           f13(x) lt -1 lc 13 lw 3, f14(x) lt -1 lc 14 lw 3,\
           f15(x) lt -1 lc 15 lw 3, f16(x) lt  0 lc  1 lw 9,\
           f17(x) lt  0 lc  2 lw 9, f18(x) lt  0 lc  3 lw 9,\
           f19(x) lt  0 lc  4 lw 9, f20(x) lt  0 lc  5 lw 9
set size 1,1; set origin 1,4; unset title; set key outside
plot [a:b] f21(x) lt -1 lc 1 lw 3, f22(x) lt -1 lc 2 lw 3,\
           f23(x) lt -1 lc 3 lw 3, f24(x) lt -1 lc 4 lw 3,\
           f25(x) lt -1 lc 5 lw 3, f26(x) lt -1 lc 0 lw 3,\
           f27(x) lt  0 lc 1 lw 6, f28(x) lt  0 lc 2 lw 6
```

```
set size 1,1; set origin 2,4              # размер и координаты рис.(1,3)
set key inside; unset title               # легенда внутри
set xtics 0,1,6
a=0; b=2*pi
plot [a:b] f29(x) lt -1 lc 3 lw 3
set size 1,1; set origin 0,3              # размер и координаты рис.(2,1)
a=0; b=2*pi
plot [a:b] f30(x) lt -1 lc 3 lw 3
set size 1,1; set origin 1,3              # размер и координаты рис.(2,2)
a=0; b=2*pi
plot [a:b] f31(x) lt -1 lc 3 lw 3
set size 1,1; set origin 2,3              # размер и координаты рис.(2,3)
set key
set xtics -1,0.5,1
a=-1.0; b=1
plot [a:b] f32(x) lt -1 lc 1 lw 9
set size 2,1; set origin 0,2              # размер и координаты рис.(3,1)
set key; set xtics 0,50,300
a=0; b=100.0*pi
plot [a:b]  f34(x) lt -1 lc 3 lw 3
set size 1,1; set origin 2,2              # размер и координаты рис.(3,2)
set key; set xtics -1,0.5,1
a=-1.0; b=1
plot [a:b]  f33(x) smooth csplines  lt -1 lc 1 lw 9
set size 1,1; set origin 0,1;             # размер и координаты рис.(4,1)
set key outside; set xtics 0,0.2,1
a=0.0; b=1.0
plot [a:b] f35(x) lt -1 lc 1 lw 3, f36(x) lt -1 lc  2 lw 3,\
           f37(x) lt -1 lc 3 lw 3, f38(x) lt -1 lc  4 lw 3,\
           f39(x) lt -1 lc 5 lw 3, f40(x) lt -1 lc  0 lw 3,\
           f41(x) lt  0 lc 7 lw 7, f42(x) lt  0 lc  8 lw 7,\
           f43(x) lt  0 lc 9 lw 7, f44(x) lt  0 lc 10 lw 7
set size 1,1; set origin 1,1              # размер и координаты рис.(4,2)
set key inside left; set xtics 0,0.2,1.0
a=0.0; b=1.0
plot [a:b]  f45(x) lt -1 lc 1 lw 3, f46(x) lt -1 lc 2 lw 3,\
            f48(x) lt -1 lc 3 lw 3
set size 1,1; set origin 2,1              # размер и координаты рис.(4,3)
set key inside left bot; set xtics 0,0.2,1.0
a=0.0; b=1.0; plot [a:b]   f47(x) lt -1 lc 3 lw 3
set size 1,1; set origin 0,0              # размер и координаты рис.(5,1)
set key top left; set xtics 0,0.2,1.0
a=0.0; b=1.0; plot [a:b]   f49(x) lt -1 lc 3 lw 3
set size 2,1; set origin 1,0              # размер и координаты рис.(5,2)
set key outside; set xtics 0,0.2,1.0
a=0.0; b=1.0; plot [a:b]   f50(x) lt -1 lc 4 lw 3
unset multiplot
reset
```

Список литературы

[1] Горелик А.М. 2018. Эволюция языка ФОРТРАН. Устаревшие черты языка и средства для их замены // Препринты ИПМ им. М.В.Келдыша. 2018. №130. 13 с.

doi:10.20948/prepr-2018-130

URL:http://library.keldysh.ru/preprint.asp?id=2018-130

https://library.keldysh.ru/preprint.asp?id=2018-130

[2] Горелик А.М. 2006. Программирование на современном Фортране. – М.: Финансы и статистика, – 352 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] Немнюгин М.А., Стесик О.Л. 2008. ФОРТРАН в задачах и примерах многопроцессорных вычислительных систем. – СПб.: БХВ-Петербург, 320 с.

[6] Numerical recipes in FORTRAN–77: The art of scientific computing (ISBN 0-521-43064-X) Copyright(C) 1986–1992 by Cambridge University Press.

[7] Numerical recipes in FORTRAN–90: The art of scientific computing (ISBN 0-521-43064-X) Copyright(C) 1986–1992 by Cambridge University Press.

[8] 1994. Лабораторный практикум по высшей математике: Учеб. пособие для втузов.–2-е изд., перераб. и доп.–М.: Высш. шк., –416 с.

[9] Крылов В.И., Шульгина Л.Т. Справочная книга по численному интегрированию, 1966. Издательство “Наука”, Гл. ред. физ.–мат. лит., – 372 с. –

[10] Крылов В.И. Приближенное вычисление интегралов, 1967. Издательство “Наука”, Гл. ред. физ.–мат. лит., – 500 с. –

[11] Пономаренко А.К. Алгол-процедуры (сборник), выпуск 12, с. 6

[12] Дж. ФОРСАЙТ, М. МАЛЬКОЛЬМ, К. МОУЛЕР, Машинные методы математических вычислений, МИР, МОСКВА 1980 СТР. 118