УДК 519.683

В. И. Усиченко, А. В. Крюков

ГП «КБ «Южное» имени М.К.Янгеля

АНАЛИЗ ОСОБЕННОСТЕЙ НЕКОТОРЫХ АЛГОРИТМОВ ВЫЧИСЛЕНИЯ ЛЕЖАНДРОВЫХ ПОЛИНОМОВ ВЫСОКИХ ПОРЯДКОВ

Розглянуті особливості різних алгоритмів обчислення поліномів Лежандра високих порядків з точки зору зручності їх програмної реалізації та швидкодії.

Ключові слова: *алгоритм, поліноми Лежандра, рекурентне співвідношення, формула Лапласа, квадратурна формула, залишковий член, факторіал, гіпергеометрична функція.*

Рассмотрены особенности различных алгоритмов вычисления полиномов Лежандра высоких порядков с точки зрения удобства их программной реализации и быстродействия .

Ключевые слова: *алгоритм, полиномы Лежандра, рекуррентное соотношение, формула Лапласа, квадратурная формула, остаточный член, факториал, гипергеометрическая функция.*

Efficacy of various algorithms for calculation of Legendre polynomials of high orders is investigated. Convenience of their programming is estimated.

Keywords: *algorithm, Legendre polynomial, recurrent relationship, Laplace formula, quadrature formula, residual member, factorial, hypergeometric function.*

Цель работы

При высокоточных расчетах компонент вектора гравитационного ускорения важен выбор быстродействующих алгоритмов вычисления полиномов и присоединенных функций Лежандра высоких (в идеале произвольных) порядков. Несложно убедиться, что, например, расчет движения КА с высокой точностью на длительных временных интервалах может потребовать значительного машинного времени. Поэтому программный код такой задачи нуждается в многоуровневой оптимизации в целом, включая и оптимизацию кода расчета вектора гравитационного ускорения в текущей точке орбиты. В свою очередь, алгоритмы расчета главных и присоединенных лежандровых полиномов и их производных имеют высокую степень вложенности в алгоритм расчета компонент гравитационного ускорения и являются его важной составной частью. Отсюда следует, что оптимизация алгоритмов для лежандровых полиномов и их производных оказывает значительное влияние на эффективность алгоритма расчета вектора гравитационного ускорения.

В связи с этим представляет практический интерес сравнительный анализ эффективности ряда алгоритмов для вычисления лежандровых полиномов высоких порядков в плане их быстродействия и удобства программной реализации. Ниже приводятся основные результаты такого анализа.

Различные формы представления лежандровых полиномов как источник алгоритмов для их вычисления

Полиномы и присоединенные функции Лежандра, как известно, получили свое название в честь французского математика XVIII-XIX веков Лежандра Андриена Мари (18.09.1752-10.01.1833). Характерно, что указанные полиномы впервые появились в решаемой Лежандром задаче об определении направленной вдоль радиус-вектора компоненты силы притяжения эллипсоида (1783, опубликовано в 1785). Полиномы Лежандра явились исторически первой системой ортогональных многочленов и за истекшие почти два с половиной столетия были хорошо изучены. Многообразие свойств и методов представления полиномов Лежандра дают возможность построения целого спектра алгоритмов их вычисления. Рассмотрим некоторые из этих алгоритмов с точки зрения удобства их программирования.

Если учесть, что наиболее распространенная форма представления лежандровых полиномов в виде формулы Родрига заведомо не пригодна для машинной реализации, то невольно обращает на себя внимание ее разновидность в виде суммы [1]

$$P_n(z)=\frac{1}{2^n}\cdot\sum_{k=0}^{E\left(\frac{n}{2}\right)}\frac{(2n-2k)!}{k!(n-k)!(n-2k)!}\cdot z^{n-2k} \qquad (1),$$

называемой еще явным выражением для многочленов Лежандра. Приведенная только что сумма пригодна для машинного вычисления полиномов Лежандра не высоких порядков. Однако для полиномов Лежандра достаточно высоких порядков (~20 и более) ее пригодность сомнительна. Во-первых, в ней содержится пять циклов, что при высоких порядках полиномов будет способствовать снижению быстродействия кода, а, во-вторых, мы сталкиваемся с проблемой вычисления факториалов больших чисел, о которой будет сказано ниже. Поэтому (1) вряд ли может быть источником высокопроизводительного алгоритма при высоких порядках лежандровых полиномов.

Имеют место также ряд интегральных представлений полиномов Лежандра. Для наших целей представляют интерес следующие [1,2]:

-интегральное представление Лапласа

$$P_n(z)=\frac{1}{\pi}\cdot\int_0^{\pi}\left(z+\sqrt{z^2-1}\cdot\cos\varphi\right)^n d\varphi; \qquad (2)$$

-интегральное представление Мейера:

$$\left.\begin{aligned} P_n(\cos\theta) &= \frac{2}{\pi}\cdot\int_0^{\theta}\frac{\cos\left[(n+0.5)\varphi\right]}{\sqrt{2\cdot(\cos\varphi-\cos\theta)}}d\varphi \\ P_n(\cos\theta) &= \frac{2}{\pi}\cdot\int_{\theta}^{\pi}\frac{\sin\left[(n+0.5)\varphi\right]}{\sqrt{2\cdot(\cos\theta-\cos\varphi)}}d\varphi \end{aligned}\right\} \qquad (3)$$

Оба соотношения в (3) следует использовать совместно, так как первое из них при θ=0, а второе при θ=π тождественно равно нулю. Так, например, при n=4 и θ=0 первое соотношение даст $P_4(1)=0$, а второе соответственно $P_4(1)=1$. Если учесть, что

$$P_4(z)=\frac{1}{8}\cdot\left(35z^4-30z^2+3\right),$$

то становится очевидным, что верный результат дает второе соотношение из (3). При θ=π и n=4 картина получается прямо противоположная, то есть теперь уже второе соотношение в (3) даст не верный результат, а первое – верный. Следовательно, при использовании интегрального представления Мейера первое равенство в (3) следует использовать для вычисления полиномов Лежандра на левом конце интервала [-1;1], а второе – на правом. Во внутренних точках отрезка [-1;1] применимы оба соотношения из (3). Впрочем, при построении алгоритмов на концах отрезка [-1;1] мы будем использовать известное соотношение

$$P_n(\pm 1)=(\pm 1)^n, \qquad (4)$$

что дает возможность для внутренних точек из [-1;1] ограничиться одним из соотношений (3). Понятно, что соотношения (2) и (3) лишены недостатков (1) и могут в принципе быть использованы для эффективного, то есть точного и быстрого, вычисления полиномов Лежандра в заданной точке отрезка [-1;1].

Известно также, что лежандровы полиномы более высоких порядков и их производные связаны с полиномами более низких порядков рядом рекуррентных соотношений:

$$(n+1)\cdot P_{n+1}(z)-(2n+1)\cdot z\cdot P_n(z)+n\cdot P_{n-1}(z)=0 \qquad (5),$$

$$\left(1-z^2\right)\cdot\frac{dP_n(z)}{dz}=n\cdot P_{n-1}(z)-n\cdot z\cdot P_n(z) \qquad (6),$$

и

$$\frac{dP_{n+1}(z)}{dz}-\frac{dP_{n-1}(z)}{dz}=(2\cdot n+1)\cdot P_n(z) \qquad (7).$$

Наконец многочлены Лежандра удовлетворяют дифференциальному уравнению Лежандра, являющимся важным аналитическим инструментом изучения сферических функций. Уравнение Лежандра имеет следующий вид

$$\left(1-z^2\right)\frac{d^2P_n(z)}{dz^2}-2z\frac{dP_n(z)}{dz}+n\cdot(n+1)\cdot P_n(z)=0 \qquad (8).$$

Существуют также другие представления полиномов Лежандра. В частности нам известны их представления на основе использования аппарата цепных дробей и с использованием гипергеометрической функции. Гипергеометрическая функция, как основа для построения алгоритма вычисления главных и присоединенных полиномов Лежандра, будет рассмотрена ниже; цепные же дроби для лежандровых полиномов, на наш взгляд, требуют отдельного рассмотрения и в данной работе не затрагиваются.

Перейдем к анализу эффективности алгоритмов, следующих из (2)-(8) и свойств гипергеометрической функции.

Особенности алгоритма вычисление полиномов Лежандра на основе их представления интегральными соотношениями Мейера

Выше уже упоминалось о возможности вычисления лежандровых полиномов на основе совместного использования интегральных соотношений Мейера (3). Простота программной реализации этого метода не вызывает сомнений. Алгоритм вычисления интеграла одним из численных методов легко поддается двухуровневой оптимизации, а его простота и, в силу этого, эффективность кажутся весьма привлекательными. Однако при этом следует учитывать тот факт, что любой численный метод вычисления интеграла является приближенным. Поэтому при выборе метода нахождения интегралов (3) следует оценить точность метода, то есть его остаточный член. В процессе анализа нами было рассмотрено два метода для реализации алгоритмов вычисления интегралов в (3): метод вычисления интегралов по формуле Симпсона и формуле Ньютона-Котеса. Известно, что для интегрируемой на отрезке [a,b] функции *f(x)* величина остаточного члена R_s в случае вычисления ее интеграла на [a,b] по формуле Симпсона задается соотношением

$$R_S = -\frac{h^5}{90}\cdot f^{IV}(\xi) \qquad (9)$$

где h-шаг интегрирования, ξ-некоторое число из отрезка [a,b]. Для остаточного члена R_{NK} в случае вычисления того же интеграла по формуле Ньютона-Котеса справедлива оценка

$$R_{NK} = -\frac{3\cdot h^5}{80}\cdot f^{IV}(\xi) \qquad (10)$$

Из сравнения (9) и (10) видим, что остаточный член для формулы Симпсона почти в три с половиной раза (3,375) меньше. Следовательно, при использовании соотношений (3) для вычисления интегралов целесообразно использовать усложненную квадратурную формулу Симпсона для интервала [0,θ] в случае использования первого из соотношений (3) и интервала [θ,π] – при использовании второго. В обоих случаях квадратурная формула Симпсона для (3) будет иметь вид

$$P_n(\cos\theta) = \frac{2h}{3\pi}\left[f(a) + f(a+2Nh) + 4\sum_{k=1}^{N} f\left(a+(2k-1)h\right) + 2\sum_{k=1}^{N} f(a+2kh)\right] \qquad (11)$$

где:

N – число содержащихся в интервале интегрирования подинтервалов;

$h = \frac{b-a}{2N}$ - шаг интегрирования;

a,b – соответственно верхний и нижний пределы интегрирования, значения которых зависят от того, какой из двух интегралов в (3) используется;

θ=arcos(z), где, в свою очередь, z∈[-1;1] и является точкой, в которой необходимо вычислить полином Лежандра.

Оценим порядок погрешности в случае вычисления по (11) первого интеграла из (3). Тогда в (11) следует положить a=0, b=θ и (n=const)

$$f(\varphi) = \frac{\cos[(n+0.5)\varphi]}{\sqrt{2\cdot(\cos\varphi - \cos\theta)}} \qquad (12)$$

Из (9) видим, что на величину остаточного члена влияет значение четвертой производной подынтегральной функции. В свою очередь из рисунка 1, содержащего графики функции (12) при n=0,1,2,3,4,5,18 и 100, и θ=π/2 видим, что первые производные функций (12) по мере приближения их аргумента к π/2 резко возрастают независимо от порядка полинома Лежандра. Анализ свидетельствует, что и модули четвертых производных функций (12) имеют тенденцию к резкому возрастанию при приближении аргумента φ к верхнему пределу интегрирования. Так при n=100 и z=0,3, что соответствует θ=1,266 в точке φ=0,14 имеем

$$\left|\frac{d^4}{d\varphi^4}\frac{\cos[(n+0.5)\varphi]}{\sqrt{2\cdot(\cos\varphi-\cos\theta)}}\Big|_{\varphi=0.14}\right| \approx 6{,}2\cdot 10^6,$$

а при z=0,99 в той же точке и при том же n получаем уже

$$\left|\frac{d^4}{d\varphi^4}\frac{\cos[(n+0.5)\varphi]}{\sqrt{2\cdot(\cos\varphi-\cos\theta)}}\Big|_{\varphi=0.14}\right| \approx 6\cdot 10^{12}$$

Таким образом, при шаге h=0,01 на концах интервала остаточный член (9) может достигать недопустимой величины

$$R_S = -\frac{10^{-10}\cdot 6\cdot 10^{12}}{90} \approx 6{,}7$$

При шаге h=0,001 эта величина будет уже равна 0,00007.

Отсюда следует, что при вычислении полиномов Лежандра на основе интегральных соотношений (3) решающее значение для точности алгоритма имеет выбранный шаг интегрирования. Значение h шага должно быть оптимизировано с точки зрения соотношения точность/быстродействие алгоритма.

Уменьшение шага интегрирования, однако, приводит к некоторому снижению быстродействия основанного на (3) алгоритма.

В целом же можно считать, что интегральные соотношения Мейера (3) представляют практический интерес, поскольку уменьшение шага интегрирования с целью обеспечения необходимой точности на краях отрезка [-1;1], при правильной организации программного кода для (11) и (12), компенсируется простотой последнего.

Вычисление лежандровых полиномов с использованием формулы Лапласа

Алгоритм вычисления полиномов Лежандра в заданной точке z [-1;1] можно также построить на основе формулы Лапласа (2). Ее применение, как и в случае с интегральными представлениями Мейера, связано с вычислением интеграла. Из (2) получаем:

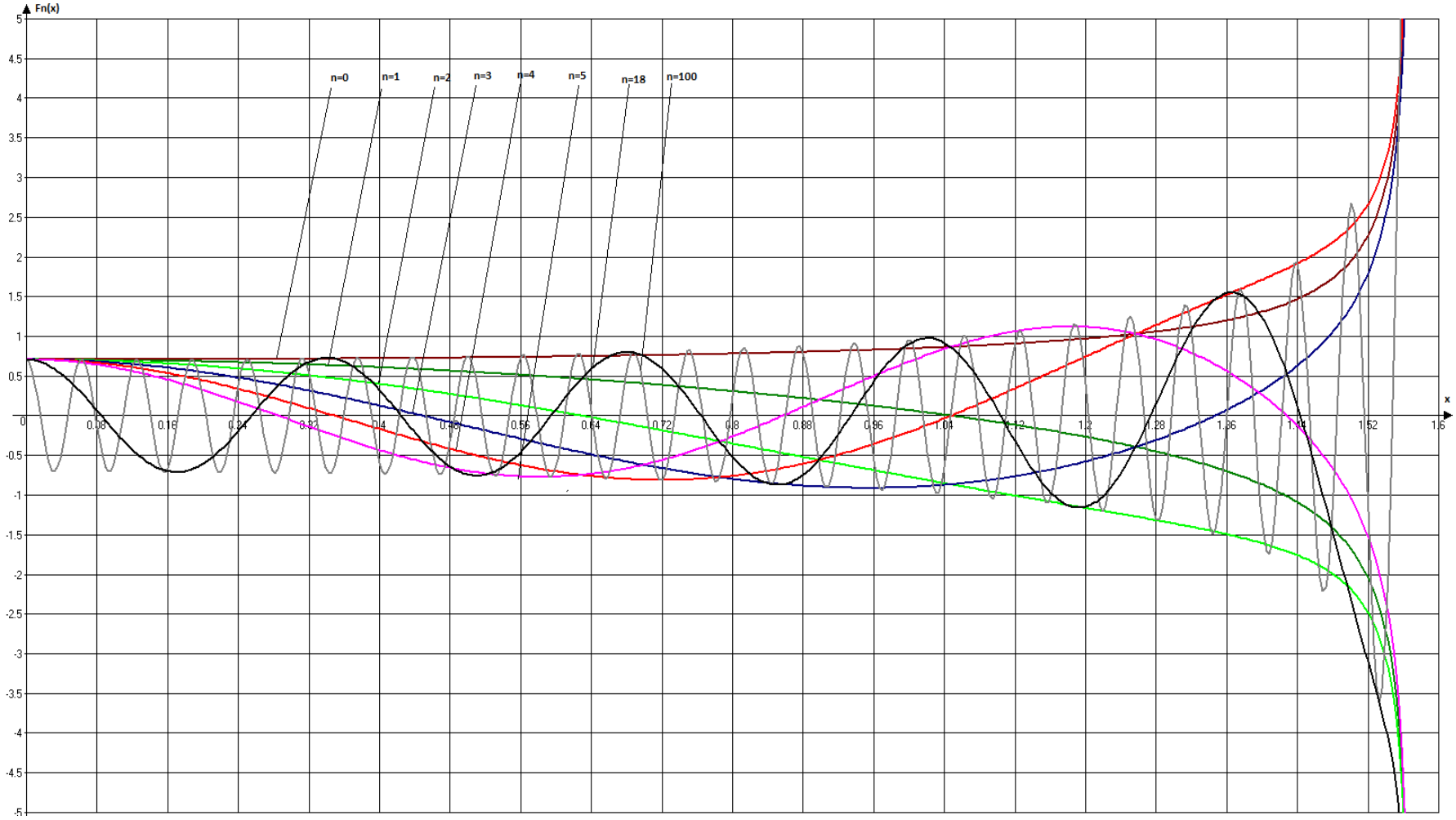

Рис. 1 Вычисление лежандровых полиномов с использованием формулы Лапласа

$$P_n(z)=\frac{1}{\pi}\cdot\int_0^{\pi}\left(z+\sqrt{z^2-1}\cdot\cos\varphi\right)^n d\varphi=\frac{1}{\pi}\cdot\int_0^{\pi}\left(z+\sqrt{i^2\left(1-z^2\right)}\cdot\cos\varphi\right)^n d\varphi,$$

то есть

$$P_n(z)=\frac{1}{\pi}\cdot\int_0^{\pi}\left(z+i\sqrt{1-z^2}\cdot\cos\varphi\right)^n d\varphi \qquad (13)$$

Последнее выражение, как известно, приводится к виду

$$P_n(z)=\sum_{k=0}^{n}(-1)^k C_n^k\cdot\frac{(2k-1)!!}{(2k)!!}\cdot\left(1-x^2\right)^k\cdot x^{(n-2k)} \qquad (14)$$

где C_n^k - биномиальные коэффициенты. Отсюда следует, что вместо нахождения (13) одним из численных методов можно прибегнуть к точному выражению (14).

Составление и реализация алгоритма нахождения интеграла (13) с применением формулы Симпсона представляет собой типовую задачу вычислительной математики. При этом для работы с комплексными числами в C++ целесообразно использовать функции *imag(val)* и *real(val)* класса *complex*< >, т.к. они позволяют напрямую обращаться к вещественной и мнимой части комплексного числа.

Что касается аналога выражения (13) в виде конечной суммы (14), то она в большинстве современных компиляторов может быть вычислена с упором на стандартные типы данных лишь для весьма умеренных значений n. В случае же очень высоких порядков лежандровых полиномов стандартные типы данных в (14) быстро переполняются. Устранение этого явления требует принятия специальных мер, которые не всегда могут быть оправданы в первую очередь с точки зрения трудозатрат на их реализацию.

Полиномы Лежандра и гипергеометрическая функция

При построении алгоритмов вычисления полиномов Лежандра следует обратить внимание на их выражение через гипергеометрическую функцию $F(\alpha,\beta,\gamma;z)$, которая является аналитическим продолжением гипергеометрического ряда и удовлетворяет линейному дифференциальному уравнению

$$z\cdot(1-z)\cdot\frac{d^2w(z)}{dz^2}+\left[\gamma-(\alpha+\beta+1)\cdot z\right]\cdot\frac{dw(z)}{dz}-\alpha\cdot\beta\cdot w(z)=0 \qquad (15)$$

Уравнение (15) называется гипергеометрическим уравнением или дифференциальным уравнением Гаусса. Гипергеометрическое уравнение рассматривается в комплексной плоскости, а его решение в символах Аппеля имеет вид

$$F(\alpha,\beta,\gamma;z)=\sum_{n=0}^{\infty}\frac{(\alpha,n)\cdot(\beta,n)}{(\gamma,n)\cdot(1,n)}\cdot z^n \qquad (16)$$

Учитывая, что для натуральных a и b символ Аппеля (a,b) выражается через Г-функцию как

$$(a,b)=\frac{\Gamma(a+b)}{\Gamma(a)}, \tag{17}$$

из (16), учитывая, что $\Gamma(k+1)=k\cdot\Gamma(k)$ несложно получить

$$F(\alpha,\beta,\gamma;z)=\sum_{n=0}^{\infty}\frac{\Gamma(\alpha+n)\cdot\Gamma(\beta+n)\cdot\Gamma(\gamma)}{\Gamma(\alpha)\cdot\Gamma(\beta)\cdot\Gamma(\gamma+n)\cdot n!}\cdot z^n, \tag{18}$$

Расходимость (18) внутри единичного круга $|z|<1$ будет иметь место в случае, когда γ есть нуль или целое отрицательное число, а также когда γ и α или γ и β оба нули или целые отрицательные числа. В остальных случаях ряд (18) сходится.

Из теории обыкновенных дифференциальных уравнений известно, что одно из решений уравнения (15) , называемое функцией Лежандра первого рода, как раз и дает выражение главных полиномов Лежандра и имеет вид

$$P_n(z)=F\left(-n,n+1,1;\frac{1-z}{2}\right) \tag{19}$$

где $z=x+iy$ есть комплексное число. Так как полиномы Лежандра при расчете вектора гравитационного ускорения берутся на отрезке [-1;1], то в нашем случае следует положить $\mathrm{Im}\, z = 0$. Тогда (19) приобретает вид

$$P_n(x)=F\left(-n,n+1,1;\frac{1-x}{2}\right) \tag{20}$$

и выражается в виде ряда

$$F(\alpha,\beta,\gamma;x)=1+\frac{\alpha\beta}{\gamma}\cdot\frac{x}{1!}+\frac{\alpha\cdot(\alpha+1)\cdot\beta\cdot(\beta+1)}{\gamma\cdot(\gamma+1)}\cdot\frac{x^2}{2!}+\frac{\alpha\cdot(\alpha+1)\cdot(\alpha+2)\cdot\beta\cdot(\beta+1)\cdot(\beta+2)}{\gamma\cdot(\gamma+1)\cdot(\gamma+2)}\cdot\frac{x^3}{3!}+\ldots \tag{21}$$

Иными словами получается, что (21) дает выражение гипергеометрического ряда (19) в области действительных чисел. А ведь именно этот случай нас и интересует, если учесть структуру сферических функций в формулах разложения в ряд потенциала земного поля тяготения.

В итоге получаем важный вывод о том, что для вычисления лежандровых полиномов на [-1;1] достаточно воспользоваться соотношениями (20) и (21). Принципиальный характер последнее утверждение приобретает в силу того, что (21) может лежать не только в основе алгоритма вычисления главных, *но и присоединенных полиномов Лежандра.* Так для присоединенной функции Лежандра произвольного порядка n и степени m имеет место соотношение:

$$P_{n,m}(x)=\frac{(-1)^m\cdot(2n)!}{2^n\cdot n!\cdot(n-m)!}\cdot\sqrt{\left(1-x^2\right)^m}\cdot x^{n-m}\cdot F\left(\frac{m-n}{2},\frac{m-n+1}{2},\frac{1}{2}-n;\frac{1}{x^2}\right) \tag{22}$$

Как видим гипергеометрическая функция (21) действительно является в некотором смысле универсальным инструментом, пригодным как для

вычисления основных полиномов Лежандра, так и лежандровых присоединенных функций. Данная ситуация позволяет создать на основе (21) некий многократно используемый код, пригодный для вычисления $F(\alpha,\beta,\gamma;z)$ как в (20) так и в (22). Использование единой функции $F(\alpha,\beta,\gamma;z)$ как для главных, так и присоединенных полиномов Лежандра обеспечивает уменьшение общего объема кода и упрощает программирование.

Если в (21) α=1, β=γ то гипергеометрическая функция дает геометрическую прогрессию:

$$F(\alpha,\beta,\gamma;x)=1+x+x^2+x^3+... \tag{23}$$

Именно этим фактом обусловлено название «гипергеометрическая» для функции (21). При |x|<1 ряд (21), как известно, сходится (бесконечно убывающая геометрическая прогрессия), при |x|>1 - расходится. При |x|=1 сходимость гипергеометрического ряда (21) зависит от значения γ-α-β, а именно:

- при x=1 ряд сходится абсолютно если γ-α-β>0 и расходится если γ-α-β≤0;
- при x=-1 абсолютно сходится если γ-α-β>0, сходится условно если -1< γ-α-β≤0 и расходится, если γ-α-β≤-1.

Если α или β равно отрицательному целому или нулю, то гипергеометрический ряд (21) становится *конечным*, то есть превращается в многочлен. Из (20) видим, что именно этот случай имеет место (α<0) при вычислении значений полиномов Лежандра на (-1;1). Это сильно упрощает применение гипергеометрической функции. Так, например, при n=2 из (20) получаем всего четыре слагаемых

$$P_2(x)=F\left(-2,3,1,\frac{1-x}{2}\right)=1-6\cdot\frac{1-x}{2}+\frac{(-2)\cdot(-1)\cdot3\cdot4}{1\cdot2}\cdot\frac{(1-x)^2}{4\cdot2!}+$$
$$+\frac{(-2)\cdot(-1)\cdot0\cdot3\cdot4\cdot5}{1\cdot2\cdot2}\cdot\frac{(1-x)^3}{8\cdot3!}=\frac{1}{2}\cdot\left(3\cdot x^2-1\right) \tag{24}$$

Наконец если γ равно отрицательному целому или нулю, то гипергеометрический ряд (21) не определен, при условии, что ни α, ни β не равны отрицательному целому (α≠-m, β≠-n причем |m|<|n|).

Структура и «минусы» алгоритмов на основе гипергеометрического ряда

Алгоритм построения ряда (21) для главных полиномов Лежандра изображен на рисунке 2. Здесь n – порядок полинома, а z точка, в которой требуется вычислить значение полинома. Алгоритм предусматривает также использование массива N [ ] вещественных чисел, элементами которого являются члены ряда (21), а размерность N [ ] равна наивысшему порядку вычисляемых полиномов.

Как уже отмечалось, код вычисления суммы (21) может использоваться при вычислении главных и присоединенных полиномов Лежандра, а также их

производных, что способствует уменьшению объема общего кода приложения. Однако, не смотря на эту его привлекательную черту, при очень высоких порядках полиномов алгоритм на базе гипергеометрического ряда (21) отягощен проблемой вычисления факториалов больших чисел.

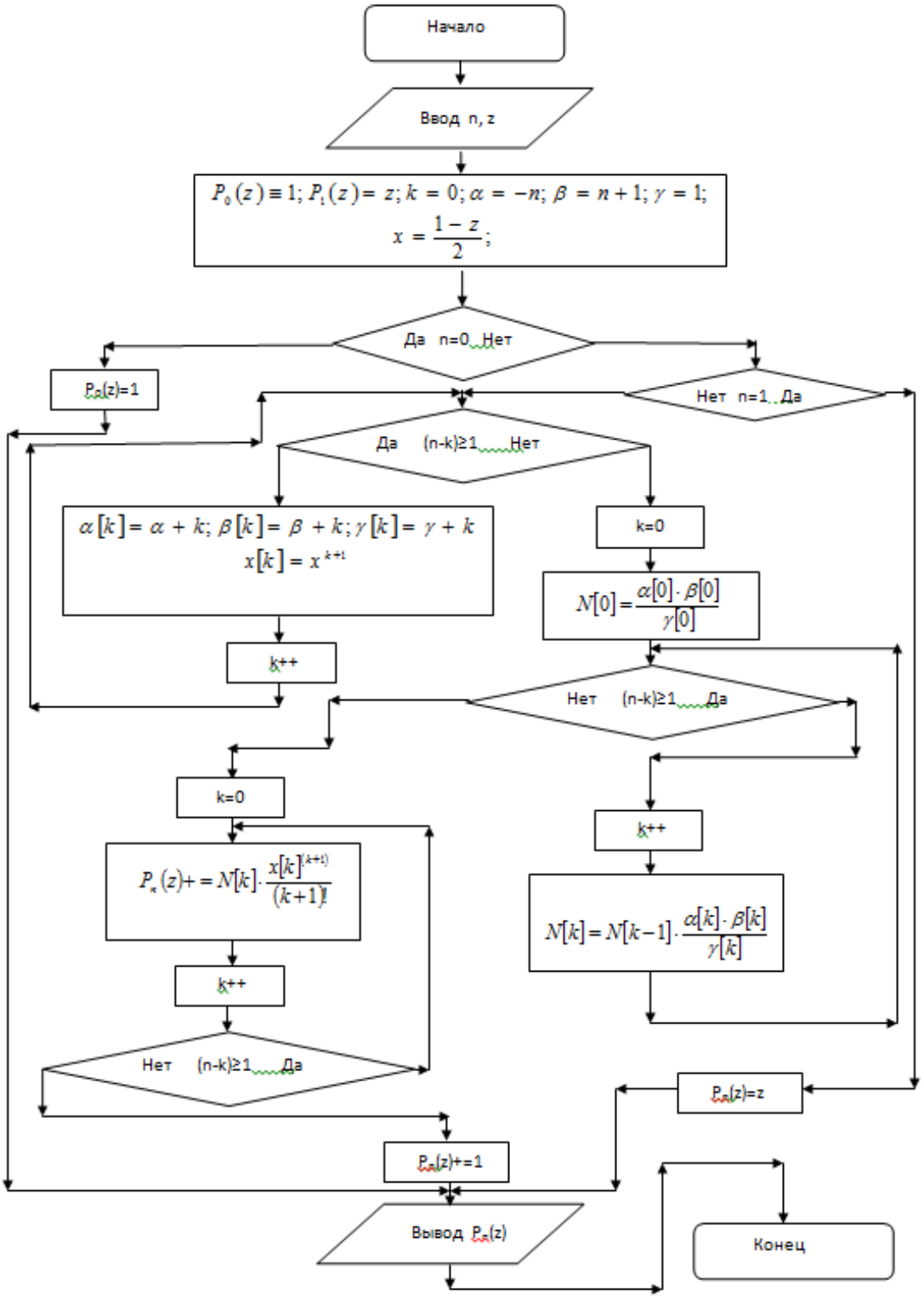

Рис. 2 Структура алгоритмов на основе гипергеометрического ряда

Сама процедура нахождения факториала в большинстве языков, как известно, реализуется предельно просто. Более того, отдельные из них имеют даже встроенную функцию для факториала. Проблема кроется в возможности представления полученных факториалов стандартными типами данных, когда порядки полиномов достигают трех и даже четырехзначных чисел. У большинства современных компиляторов стандартные целые типы переполняются значениями факториалов уже на третьем десятке порядков лежандровых полиномов. Так, например, двадцать шестой член в (21) в знаменателе будет содержать 25 != 15511210043330985984000000. Что же говорить о более высоких порядках? Ведь в некоторых современных гравитационных моделях Земли количество коэффициентов разложения потенциала в ряд по сферическим функциям превышает 2000, а это предусматривает вычисление присоединенных функций Лежандра соответствующих порядков и степеней, а также их производных.

Число цифр c(N) в факториале натурального числа N очень быстро растет с ростом самого N. Приведем лишь в качестве иллюстрации некоторые примеры: c(0)=C(1)=1, c(5)=3,c(10)=7, c(25)=26, c(50)=65, c(86)=132, c(150)=264, и далее c(500)=1122,c(1000)=2565. Рост числа разрядов наглядно можно проиллюстрировать на примере факториала тысячи. Для указанной величины получаем (нами вычислено в среде UBASIC 8.30):

1000!=4023872600770937735437024339230039857193748642107146325437999104299385123986290205920442084869694048004799886101971960586316668729948085589013238296699445909974245040870737599188236277271887325197795059509952761208749754624970436014182780946464962910563938874378864873371191810458257836478499770124766328898359557354325131853239584630755574091142624174743493475534286465766116677973966688202912073791438537195882498081268678383745597317461360853795345242215865932019280908782973084313928444032812315586110369768013573042161687476096758713483120254785893207671691324484262361314125087802080002616831510273418279777047846358681701643650241536913982812648102130927612448963599287051149649754199093422215668325720808213331861168115536158365469840467089756029009505376164758477284218896796462449451607653534081989013854424879849599533191017233555566021394503997362807501378376153071277619268490343526252000158885351473316117021039681759215109077880193931781141945452572238655414610628921879602238389714760885062768629671466746975629112340824392081601537808898939645182632436716167621791689097799119037540312746222899880051954444142820121873617459926429565817466283029555702990243241531816172104658320367869061172601587835207515162842255402651704833042261439742869330616908979684825901254583271682 2

64580665267699586526822728070757813918581788896522081643483
44825993266043367660176999612831860788386150279465955131156
55203609398818061213855860030143569452722420634463179746059
46825731037900840244324384656572450144028218852524709351906
20929023136493273497565513958720559654228749774011413346962
71542284586237738753823048386568897646192738381490014076731
04466402598994902222217659043399018860185665264850617997023
56193897017860040811889729918311021171229845901641921068884
38712185564612496079872290851929681937238864261483965738229
11231250241866493531439701374285319266498753372189406942814
34118520158014123344828015051399694290153483077644569099073
15243327828826986460278986432113908350621709500259738986355
42771967428222487575867657523442202075736305694988250879689
28162753848863396909959826280956121450994871701244516461260
37902930912088908694202851064018215439945715680594187274899
80942547421735824010636774045957417851608292301353580818400
96996372524230560855903700624271243416909004153690105933983
83577793941097002775347200000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000.

Между тем, если говорить о применимости гипергеометрического ряда (21) для вычисления лежандровых полиномов достаточно высоких (читай произвольных) порядков, в вопросе вычислимости факториалов необходимо иметь ясность. Для вычисления полиномов Лежандра высоких порядков с использованием гипергеометрического ряда существует, на наш взгляд, два метода.

Первый из них заключается в использовании целочисленной арифметики повышенной точности. Реализация этого метода возможна лишь на основе программирования на языке ассемблера, который позволяет создавать типы данных произвольного размера. Например, можно для чисел выделить 1024 бита, то есть 32 32-битных слова (32^2=1024). Так, например, строка с кодом объявления 96-битного числа *longnumber* будет иметь вид

```
Longnumber  dw 0,0,0      ;32*3=96
```

Соответственно для отвода под число longnumber одного килобайта (1024 бит) в только что приведенной строке понадобится уже тридцать два нуля. Понятно, что такой подход в принципе решает проблему применимости гипергеометрического ряда для вычисления полиномов Лежандра в рамках абсолютного большинства современных гравитационных моделей Земли. Неудобство введения специальных типов данных для целочисленной

арифметики заключается в том, что сложность кодирования арифметических операций над такими числами растет пропорционально количеству отведенных под них слов. Особенно это касается операций деления и умножения, которые являются актуальными при построении гипергеометрического ряда. Для операций сложения и вычитания ситуация несколько проще, так как ассемблерные команды ADC (сложение со сдвигом) и SBB (вычитание с учетом займа) были специально введены для целочисленной арифметики повышенной точности.

Из сказанного следует, что реализация алгоритмов для вычисления лежандровых полиномов очень высоких порядков на основе гипергеометрического ряда может оказаться *весьма* трудоемкой. Ее кодирование на ассемблере по трудозатратам может оказаться соизмеримым с трудозатратами на кодирование основной задачи, в которой используются полиномы Лежандра высоких порядков.

Второй способ обеспечения использования гипергеометрического ряда при вычислении полиномов Лежандра высоких порядков может заключаться в написании соответствующих программных модулей на одном из специализированных языков *высокого уровня*. Другими словами модуль реализации алгоритма, блок-схема которого приведена на рисунке 2, целесообразно написать на языке, заведомо поддерживающем целочисленную арифметику высокой точности. Из известных нам языков программирования таковым является, например, UBASIC 8.30. Данный язык программирования высокого уровня был создан профессором Юджи Кида на математическом отделении университета в Риккио (Япония) и предназначен в первую очередь для теоретико-числовых исследований. Он сочетает простоту синтаксиса языков семейства Basic с уникальными вычислительными возможностями. В интересующем нас аспекте он вполне подходит для вычисления полиномов Лежандра высоких порядков. Достаточно сказать, что диапазон поддерживаемых в UBASIC по умолчанию целых чисел равняется $\pm 65536 \cdot 10^{542}$. Это означает, что данный язык программирования может легко работать с целыми числами, содержащими более чем 2600 цифр в десятичном представлении. Именно в среде программирования UBASIC нами были получены приведенные выше значения числа цифр в факториалах и значение 1000!. Под вещественные числа в UBASIC 8.30 отводится 542 машинных слова.

Однако, несмотря на эффективную поддержку арифметики повышенной точности языком программирования UBASIC, включение написанных на нем программных модулей в основной проект при сборке не представляется возможным, поскольку UBASIC является транслятором и не создает объектные и исполняемые модули. Компиляторы же других версий BASICa модули UBASIC не обрабатывают. Использование вычислительных возможностей UBASIC 8.30 возможно, таким образом, только в случае написания *всей задачи* в среде UBASIC 8.30, что представляется далеко не всегда рациональным из-за его незначительного распространения и некоторых других специфических черт.

Проведенный анализ приводит к выводу, что вычисление полиномов Лежандра по алгоритму, использующему соотношения (20) и (21) в любом случае ведет к снижению объема кода всего приложения и вполне оправдано при умеренных порядках полиномов, когда для представления результатов достаточно стандартных типов данных компилятора. При высоких порядках полиномов к гипергеометрическому ряду (21) и целочисленной арифметике повышенной точности следует прибегать лишь в тех случаях, когда трудозатраты на это оправдывают себя по тем или иным соображениям.

Алгоритмы на основе рекуррентных соотношений

Из указанных в начале способов представления многочленов Лежандра остались не рассмотренными алгоритмы, следующие из соотношений (5) - (7). Понятно, что ввиду отсутствия производных интерес для программной реализации может вызывать соотношение (5), откуда легко получаем

$$P_{n+1}(z)=\frac{1}{n+1}\cdot\left[(2n+1)zP_n(z)-nP_{n-1}(z)\right] \qquad (25)$$

В точке z=0 (25) принимает вид

$$P_{n+1}(0)=\frac{-nP_{n-1}(0)}{n+1} \qquad (26),$$

а с учетом некоторых свойств Лежандровых полиномов и того, что $P_0(z)\equiv 1,\quad P_1(z)=z$ легко построить алгоритм для полиномов Лежандра второго и выше порядка с использованием (25) и (26). Причем, как показывает анализ применение обилия свойств полиномов Лежандра в алгоритме *вредно*. В этом случае множество условных переходов в сочетании с циклами и высокими порядками полиномов отнюдь не способствует быстродействию. Отсюда следует вывод, что алгоритм должен быть по возможности универсальным и «не размениваться» на всякие частные случаи.

В соответствии с этим требованием получаем простую, с одним условным переходом блок-схему непосредственного вычисления по (25) лежандровых многочленов любого порядка. Эта блок-схема приведена на рисунке 3. Как видим алгоритм на основе рекуррентного соотношения (5) простой, компактный, точный и к тому же не требует выхода за пределы стандартных типов данных. Поэтому его эффективность для вычисления главных лежандровых полиномов высоких порядков вне сомнения. Это подтверждают результаты его тестирования и хронометража средствами ОС для полиномов вплоть до стотысячного порядка.

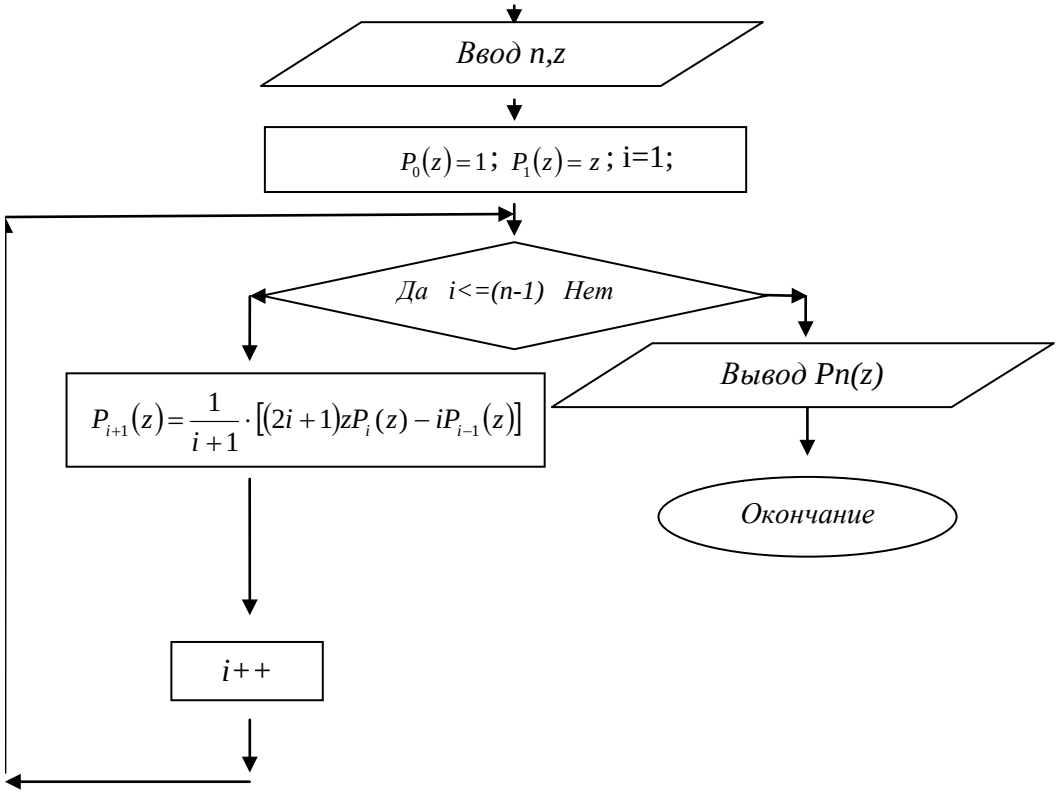

Рис. 3 Алгоритмы на основе рекуррентных соотношений

Применение соотношения (25) возможно только для главных полиномов Лежандра. Для присоединенных надо писать уже новый код. Здесь нет той инвариантной составляющей программного кода, которая присуща алгоритму на основе гипергеометрической функции и это, пожалуй, единственный, хотя и не существенный недостаток рекуррентного алгоритма. Легкость работы с любыми порядками чаще всего оправдывает реализацию в коде именно этого алгоритма.

Выводы

1. Если в расчете вектора гравитационного поля Земли используется 15-18 гармоник , когда максимальный порядок главного полинома Лежандра n_{max} позволяет вычислить n_{max} ! не выходя за пределы стандартных целых типов компилятора, то целесообразно использовать алгоритм вычисления главных лежандровых полиномов на основе гипергеометрической функции (21), так как в этом случае она может быть использована вторично при вычислении присоединенных полиномов Лежандра.

2. Если приоритетом является высокая точность вычислений и, следовательно, возникает необходимость вычисления главных лежандровых полиномов высоких порядков, то наиболее целесообразно применение рекуррентного соотношения (25). Впрочем, его применение оправдано

практически всегда, когда не стоит остро задача минимизации исходного кода программы. Это обусловлено простотой рекуррентного алгоритма.

3. Применение формулы Лапласа (2) для достаточно высоких порядков дает хорошую точность при правильно выбранном (достаточно малом) шаге интегрирования. Но достаточно малый шаг может несколько снизить быстродействие кода по сравнению с применением рекуррентных соотношений. Однако сам код алгоритма получается компактным и простым. Для интегральных представлений Мейера (3) приходим примерно к тем же выводам, хотя шаг интегрирования в этом случае должен, как правило, быть несколько меньше чем этого требует формула Лапласа для обеспечения такой же точности на концах интервала [-1;1].

Библиографические ссылки

1. Дубошин Г. Н. Небесная механика. Основные задачи и методы [Текст]/Г. Н. Дубошин.- Москва: Наука, 1968. - 799с.

2. Справочное руководство по небесной механике и астродинамике. [Текст]/Под ред. Г. Н. Дубошина.- Москва: Наука, 1968. - 584с.

3. Бордовицына Т. В. Теория движения искусственных спутников Земли. Аналитические и численные методы[Текст]/Т. В. Бородовицына, В. А. Авдюшев. – Томск: Издательство Томского университета, 2007. – 175с.

4. Двайт Г. Б. Таблицы интегралов и другие математические формулы. [Текст]/Г. Б. Двайт-Москва: Наука, 1973. - 286с.

5. Хортрон Айвор. Visual C++ 2010. Полный курс. [Текст]/Айвор Хортон- Москва-Санкт-Петербург-Киев: Диалектика, 2011. - 1206с.

6. Джосаттис Николаи М. Стандартная библиотека C++. Справочное руководство. [Текст]/Николаи М. Джосаттис- Москва-Санкт-Петербург-Киев: Диалектика, 2014. - 1129с.

7. Шилдт Герберт.Справочник программиста по C/C++.[Текст]/Герберт Шилдт.- Москва-Санкт-Петербург-Киев:Вильямс, 2003. - 344с.

8. Пирогов Владислав. Ассемблер для Windows. [Текст]/Владислав Пирогов.- Санкт-Петербург: БХВ-Петербург, 2007. - 895с.

Надійшла до редколегії 12.08.2017