БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ
ИНСТИТУТ ПО ИНФОРМАЦИОННИ И КОМУНИКАЦИОННИ ТЕХНОЛОГИИ

МЕТОДИ И СРЕДСТВА ЗА ПОДОБРЯВАНЕ НА ПРЕСМЯТАНЕТО С ВИСОКА ТОЧНОСТ НА НЯКОИ КЛАСОВЕ ЗАДАЧИ

Величко Георгиев Джамбов

Дисертационен труд, представен за присъждане
на образователна и научна степен „доктор” по докторантска
програма 01.01.12 „Информатика”, професионално направление
4.6 “Информатика и компютърни науки”

Научен консултант: акад. Васил Сгурев

София, 2016 г.

Съдържание

Увод

Предистория и мотиви за създаването на тази програмна система.

Почти всички съвременни компютърни системи реализират стандарта IEEE за 64-битова за аритметика с плаваща запетая, който предоставя 53-битова мантиса, което е точност от около 15-16 десетични знака. За повечето научни приложения тази точност е достатъчна, но има и изключения [153]. Някои от случаите в които пресмятания с висока точност са полезни:

Лошо обусловени линейни системи: често срещан сценарий, възникващ в хода на решаването на някаква по-обща задача, при който 64-битовата точност е причина за пораждане на грешки.

Голямо количество сумирания: получаване на аномалии в резултата, дължащ се на загуба на асоциативност при сумиране, например в случай на паралелна обработка, където редът на сумиране не може да се контролира [171].

Дълги симулации: разнообразни видове симулации на физически процеси, осъществявани за множество времеви интервали, в крайна сметка се отклоняват от реалното поведение, поради натрупване на грешка от закръгляване, в допълнение към грешките от дискретизация на времето и пространството.

Широко-мащабни симулации: пресмятания, които се извършват коректно за задачи с умерена размерност на еднопроцесорни системи могат да предизвикат значителни числени грешки, когато се мащабират в паралелни системи.

Анализиране на феномени в много малък мащаб: често възниква необходимост от използване на много фина разделителна способност за да се "мащабират" подобни явления.

Пресмятания на "експерименталната математика": много от съвременните резултати в експерименталната математика не могат да бъдат получени освен чрез пресмятания с много висока точност.

Тези "изключения" обаче, са твърде важни, за да бъдат пренебрегнати и изискват съответните инструменти, позволяващи осъществяването на пресмятания с висока точност, желателно от високо ниво, за да се улесни прилагането им в различни програмни среди. Последното бе и един от мотивите, довели до идеята за използване на такива инструменти в контекста на .NET Framework в операционна система Windows. Тук са необходими някои пояснения. Формално в Windows могат да се използват вече разработени библиотеки за пресмятания с висока точност. Това обаче често на практика става с някакъв вид емулация на Unix-like среда. Освен това, почти без изключения (за изключението, малко по-нататък), предвиденият интерфейс от високо ниво включва C/C++ плюс множество други езици от високо ниво, но не и C# - основният език на .NET Framework. Някои от най-известните примери за специализиран софтуер за пресмятания с висока точност:

ARPREC: пакетът включва програми за осъществяване на аритметика с произволна точност, включително много алгебрични и специални (трансцендентни) функции. Поддържа типове данни за цели, реални и комплексни числа. Предвиден е интерфейс за C++ и Fortran-90.

GMP: пакетът включва обширна библиотека от програми за пресмятания с висока точност за цели числа, рационални числа и числа с плаваща запетая. Разпространява се с GNU лиценз от Free Software Foundation и е достъпен на адрес http://gmplib.org. Предвиден е интерфейс за C++ и още 23 езика от високо.

MPFR: това е библиотека от програми на C за пресмятания с повишена точност за числа с плаваща запетая с точно закръгляване и е базирана на библиотеката на GMP. Достъпна на адрес http://www.mpfr.org.

MPFR++: MPFR с интерфейс за C++.

MPFUN90: пакет, подобен на ARPREC по отношение на функционалността от потребителска гледна точка, но написан изцяло на Fortran-90. Предвиден е интерфейс за Fortran-90.

QD: пакетът включва специални програми за аритметика "double-double" (приблизително 31 десетични знака) и "quad-double" (приблизително 62 десетични знака). Налични са интерфейси за C++ и Fortran-90 с поддръжка на типове данни за цели, реални и комплексни числа. Софтуерът е значително по-бърз ако изискваната точност е само 31 или 62 десетични знака.

Обяснението за липса на интерфейс към C# е очевидно. В общността на създателите на научен софтуер комерсиалните среди не са на почит. Погледнато от друг ъгъл обаче, Windows е много широко разпространена (дори доминираща) операционна система за настолните и преносими персонални компютри. Всеки достатъчно опитен потребител може да си инсталира и друга операционна система, разбира се. Но защо да не се даде възможност на заинтересованите от някакви задачи, изискващи пресмятания с висока точност, възможността да създават свой собствен софтуер непосредствено в .NET Framework? Именно да създават, а не да са потребители на някаква готова система. Според нас, това носи някои големи предимства. На първо място, собственият софтуер отчита цялата специфика на решаваната задача без да се ограничава само до комбинация от възможностите на готово използвана система. На второ място, не и по значение, това позволява в самия процес на проектиране и разработка да се използва много богатата "инфраструктура" на .NET Framework, покриваща изключително широк спектър теми - много повече, отколкото в която и да е готова система. Това, разбира се, отново от гледна точка на създател на софтуер, а не само на потребител. На трето място, такъв подход дава възможност за използване на вече натрупаните знания и опит при разработка в позната среда. Настоящата работа е един вариант по създаването на програмна система, която подпомага създаването на именно такъв софтуер - в средата, използвана от огромно количество потребители.

Представяната програмна система *не е* резултат на предварително изготвен план, отчитащ всички евентуално важни области. Това е просто невъзможно, като се има предвид за какъв обем работа става дума. Тя е създадена по-скоро като достатъчна база, която може да бъде надстроявана в различни посоки около ядро, включващо абсолютно необходимото. Създадените около това ядро конкретни програмни инструменти са до голяма степен избор, продиктуван от лични интереси и предпочитания.

Самата идея за създаване на такава програмна система има своята предистория. Всеки, който се опитвал да напише софтуер от ниско ниво, осъществяващ пресмятания с висока точност, знае колко трудоемка задача е това. Изисква се подобен личен опит за да се оцени истински работата на екипите, създали средствата, които след това всички

безплатно ползват. Преди няколко години, във връзка с интересите, продиктувани от решаване на задача, която е модел на система, чието поведение води до така наречения "детерминиран хаос", бяхме впечатлени от възможностите предоставяни от Python за пресмятания с произволна точност (пакетът mpmath). Един опит за възползване е [50]. Python, разбира се, е прекрасно средство за създаване на научен софтуер (особено като се използват разширения от тип gmpy). Остана обаче известно чувство на неудовлетвореност. Все пак съществува ограничението "потребител". Потребител на създадените от други средства. Във връзка с предишна дейност (опит за модернизация на предишен, писан на C, софтуер за нелинейна оптимизация) вече имахме представа за възможностите на C# и .NET Framework. Трябваше да се преодолеят естествено и колебания, отчасти свързани с предубеждения, отчасти с предварителната оценка за обема на работата, която предстои във връзка с писането на необходим минимум (включващ реализация за пресмятането на елементарни и специални функции, но не само). Впрочем, в базовата библиотека на създадената програмна система, основното пространство от имена е MPMath - в чест на съответния пакет в Python.

Някои технически подробности по изграждането на програмната система.

За реалното осъществяване на идеята от решаващо значение се оказа възможността за използване на връзка към C# на библиотека за пресмятания с произволна точност - в случая MPIR [154] (разклонение на GMP). По същество връзката изисква наличие на C# файл-обвивка (в случая xmpir.cs) с описание на сигнатурите на извикваните функции (от към C#) и средства за адресирането на извикванията от библиотеки за динамично свързване (dll) плюс самите библиотеки за динамично свързване, съдържащи функциите на MPIR. Това става на базата на вече подготвени статични библиотеки (C интерфейс). Тази задача е решена от Sergey Bochkanov през 2010 година и всичко необходимо по генерирането на тези динамични библиотеки може да се изтегли от [146]. Реализирането на описваната програмна система не би се случило, ако нямаше тази възможност. В архивния файл, изтеглен от [146], се съдържат съответните инструкции по генериране на необходимите файлове, заедно с изходните статични библиотеки, съдържащи функциите на MPIR.

На практика по-голямата част от програмната система използва този първоначален вариант на връзката от 2010 година (към MPIR от същото време). За съжаление, нова версия към настоящия момент няма, а последната версия на MPIR (2.7.0 от 26-ти юни 2015 година) вече се появи след повече от 4 години прекъсване. В тази нова версия обаче, покрай всички усъвършенствания има и маса промени в сигнатурите на функции (както и замяна на някои остарели), което прави праволинейния подход (генериране на статичните библиотеки на MPIR + съществуващата процедура, описана в [146]) невъзможен. Голямо беше обаче желанието да се извлече всичко възможно като ефективност, поне в онази част на програмната система, където бе въведена паралелна обработка с цел максимално бързодействие (вж. съответните раздели за главите "Методи и средства за подобряване численото пресмятане на някои класове определени интеграли" и главата "Бързо пресмятане на математически константи с висока точност" по-нататък в този увод). Желанието става непреодолимо, като се има предвид съществуващата възможност за генериране на изходните статични библиотеки по начин, отчитащ дори конкретния вид на използвания процесор. Това наложи да се влезе в детайлите на процедурата по генериране

на динамичните библиотеки и да се променят съответните файлове xmpir.c и xmpir.cs, така че да съответстват на заглавния файл mpir.h, получаван при генериране на статичната библиотека. Резултатът оправда усилията. Бързодействието се повиши значително.

За някои специфични изисквания и възприетия стил на изложение.

Като за всяка програмна система за математически пресмятания, основно изискване е подходящ избор на използваните математически методи/алгоритми, плюс адекватното им практическо осъществяване на ниво програмиране. Въпреки очевидността си, това изискване не е просто да се изпълни, когато става дума за програмна система с претенции за малко по-голяма всеобхватност. Конкретен пример е реализацията на специалните и елементарните математически функции, които са в ядрото на подобна система. Като оставим настрана различията в пресмятанията на една и съща функция, в различен диапазон на променливата, продиктувани от специфично асимптотично поведение в конкретни случаи, на места това би могло да се прави с друга цел. Например от множество алтернативни възможности може да се избере понякога методът, който теоретически е с най-добро поведение. Дава най-добра асимптотика за нарастване на времето на пресмятането спрямо използваната точност (в брой десетични знаци или битове). На всички им е известен обаче проблемът със скритите константи в теоретичните оценки. Очевидно O(n*ln(n)) е по-добро поведение от $O(n^2)$, но то е валидно за граничния случай и всъщност уловката в това записване е, че е съкращение на гранични неравенства, в дясната част на които присъстват различни константи - много често, неуточнени. Напълно възможно е методът предлагащ по-лошата теоретична оценка да е практически по-ефективен за пресмятания до към 400-500 знака например - различно в конкретните случаи - и тази граница може да се установи само експериментално. Този проблем съществува и на най-ниско ниво - това, осъществяващо основните аритметични операции. Там той е решен изключително ефективно от създателите на използваната библиотека (в случая MPIR, любопитните могат да видят в ръководството какви са праговете за смяна на конкретните алгоритми при умножение, да речем). Този достатъчно тривиален пример е приведен тук, за да посочи конкретен случай, в който проблемът с избора изисква и тестване. Това е още по-очевидно за случаи, в които има избор между методи с еднаква теоретична оценка. Освен това, трябва да се отчитат оценките за ефективност не само по време, но и по използвана памет.

Една допълнителна възможност в борбата за ефективност е и в използването (когато е възможно) на методи, при които основната част от пресмятанията се правят с целочислена аритметика. Тук очевидно бързодействие се печели от факта, че тя се реализира много по-бързо в компютърните системи. За съжаление, тази стратегия не винаги е приложима.

Още една възможност за повишаване на ефективността, при това изключително актуална, е използването на паралелни пресмятания. Тук обаче във всеки конкретен случай нещата зависят от много фактори. На първо място, разбира се, за дадената задача трябва да има известен алгоритъм, който е проектиран за паралелни пресмятания или поне да има очевидна възможност да бъде видоизменен по такъв начин. Освен това разпаралеляването изисква оценка на контекста, в който се прави (за използваната изчислителна среда). Например, ако дадена задача се решава паралелно като вложена задача на друга, която също се изпълнява паралелно, не се печели кой знае какво, ако изобщо нещо се печели, при условие, че и в двата случая е зададено максимално използваните ядра на процесора,

например. Самият начин, по който се извършва разпаралеляването също има значение. Тук вече съществуват множество известни схеми (вж. например [42] за контекста на използваната от нас среда). На практика известните схеми са конкретизации на важни ситуации, вариращи между двата гранични случая: паралелност по данни и паралелност по задачи, съответно независими данни и еднотипна операция, която може да се осъществи в паралелен цикъл, или отделни независими задачи, осъществявани паралелно до момента, в който се осъществява някаква синхронизация (изискват се всички резултати от задачите, за да продължат пресмятанията по-нататък). Известна демонстрация за получената ефективност с използване на паралелни пресмятания има в глава 5. "Бързо пресмятане на математически константи с висока точност" и раздел 2.4. "Използване на многоядрената архитектура" в глава 2 "Методи и средства за подобряване числеността пресмятане на някои класове определени интеграли с висока точност". Пренасянето на някои от получените там резултати в ядрото на системата и допълнителните корекции в програмните инструменти, които да се видоизменят с цел използване на паралелни пресмятания навсякъде, където е възможно, е процес, който не е завършен.

Обсъждането до тук касае само един аспект от изискванията - този, свързан с ефективност. На практика обаче, при създаване на подобна система, съществуват и други критерии. Ясна схема на реализация за всяка определена задача, по възможност несвързана с прекалено много спомагателни задачи, т.е. лекота на поддръжка. Облекчена възможност за използване на завършените решения в по-общ контекст като отделни изчислителни блокове, т.е универсалност. Последната е от много важно практическо значение поради факта, че позволява елегантна замяна на един блок с друг на даден етап в еволюцията на системата, например, когато е намерен по-подходящ метод за някаква задача (а и при самото тестване за това дали е подходящ).

Последната част от изискванията, които ще бъдат разгледани, са свързани с удобството за използване. Тук нещата са доста субективни, в зависимост от изискванията и навиците на потребителя, както и по какъв начин всъщност ще ползва системата. Има и някои безспорни (според нас) неща, които могат да се направят за удобството на използване, свързвани с възможността за лесно въвеждане на задача непосредствено в текстов вид и евентуална визуализация. На тях е отделено място в раздела " Помощни средства за изграждане на програмни инструменти" в тази уводна глава.

Във връзка с това, че изложените до момента изисквания са в противоречие помежду си (в смисъла на многокритериална задача), изводът е почти еднозначен. Реализацията зависи от предпочитанията, вкусовете, та дори и предубежденията на реализиращия (а предубежденията обикновено са в следствие на недостатъчна информираност). Освен това, тези неща също се променят с течение на времето. С натрупването на опит, променящ вижданията ни, например. Това не е някакъв опит за предварително оправдание на съществуващи несъвършенства (неизбежни), а просто констатация. Колкото и да е свързана тематиката, обвързана с реализация на програмна система за някакви математически пресмятания, с точна информация, самата реализация е неизбежно повлияна от известен субективизъм. Това, разбира се, засяга и начина на изложение, избран тук. Тъй като подобна система е синтез на различни по същество теми, подходът, включващ изчерпателност във всичко, не е практически приложим. Направен е опит да се даде възможно най-точна представа какво представлява тази система. Нивото на подробността на описанието обаче на места е доста различно. Задължително се стараем да изясним неща, които са интересни (според нас) или нетипични. Понякога това се отнася за някакъв подход или метод, без разбирането на който, просто не е ясно защо нещата

работят, ако са гледани просто като програмен код. В други случаи има фрагменти от програмен код, които касаят нетипичен подход или са специфични за използваната среда и по-рядко, като кратка възможна илюстрация. Има и важни според нас места, които заслужават някакви коментари или допълнителна информация, свързани с разглеждан въпрос, но са извън основната нишка на изложението. Например интересна възможност за използване в някаква област или нещо, което също може да се използва и е интересно, но не е използвано поради възприетия "минималистки" принцип на сегашния етап на разработката. За някои от тях е са предвидени отделни глави с приложения, други са описани в общото изложение. Възможностите за разширяване на темите покривани от програмните инструменти или усъвършенстването на отделни части на инструменталните средства системата също са събрани и систематизирани на отделно място, въпреки че някои от тези неща се споменават и преди това. В по-голямата си част изложението е групирано в глави около отделни теми, свързани с конкретни задачи на числения анализ, описващи използваните в специално създадените програмни средства методи. Макар да включват понякога препратки една към друга, те са общо взето самостоятелни. Част от материала в отделните теми, преценен като съвсем конкретен (фрагменти от програмен код) или допълнителен, е понякога изнесен в отделни раздели или главите с приложения накрая. Общата структура на изложението може да се види от съдържанието. Някои по-общи бележки, касаещи преди всичко история, методология, контекст и перспективи, свързани с темите, засегнати в отделните глави, са включени по-нататък в тази уводна глава, за да не се прекъсва твърде много основното изложение, касаещо преди всичко въпроси от типа "как се пресмята ...".

Накрая една бележка за терминологията. На места се дава успоредно термин на английски. Термини като bisection и binary splitting например, биха звучали сходно в буквален превод, макар че се отнасят за различни неща. Съответно "метод на разполовяването" (макар да се среща в литературата и като "бисекция") и "метод на двоичното разделяне", но вероятно в подобни ситуации повечето читатели биха се ориентирали по-добре, ако има и англоезичен термин.

Структура на програмната система.

В ядрото на програмната система, освен необходимите файлове за връзка, влизат файловете funcs.cs, funcs_c.cs, mpir_parse_fx.cs. Файловете за връзка включват xmpir.cs, dl4windows.dll, xmpir32.dll, xmpir64.dll и за тях вече бе споменато по-горе. Тук можем да добавим, че наличието на два варианта динамични библиотека (за 64- битов и 32-битов) прави възможно пренасянето на вече готова програма, ако при компилацията не е задаван специален аргумент за целевата платформа (в който случай се генерира по-универсален, но и по-бавен код с аргумент по подразбиране anycpu32preferred) от типа /platform:64. В funcs.cs са включени математически функции (в пространство от имена MPMath) плюс някои специални обслужващи класове (например извеждане на числа със закръгляване до задавана точност) и такива, свързани с някои математически методи. Голяма част от математическите функции са реализирани и в комплексен вариант. funcs_c.cs включва комплексната реализация, заедно със съответната поддръжка за комплексни числа с произволна точност. В mpir_parse_fx.cs са включени средства за използване на интерпретатор на математически изрази. Изградените програмни инструменти за решаване на конкретен клас задачи добавят своите файлове. По такъв начин, теоретично е възможно всичко създадено да се поддържа на едно място. На практика за всеки отделен програмен инструмент се съдържа отделна директория с копия на основните файлове,

които също могат да се променят от време на време. Така е по-удобно, макар да носи риска за поява на прекалено много версии.

Тук му е мястото да споменем, че по принцип всичко по изграждането на готови програми може да се свърши със средства, които са безплатни. Компилаторът на C#, плюс евентуално някаква Express версия на Visual Studio. Ако ще се използва нещо по-специално, например интерфейс с Windows Presentation Foundation, включващ .xaml файлове, може да се използва msbuild директно. При нас това е само на едно място и то по-скоро като експеримент, а не по необходимост. Ако ще се използва новата възможност за генериране на библиотеките на MPIR, включваща допълнителна настройка в зависимост от типа на процесора, са нужни някои допълнителни неща, също безплатни, впрочем. Например за използването на скрипта на Brian Gladman за настройка на конфигурацията на MPIR с Visual Studio, е необходим Python, за генериране на оптимизиран код е необходим асемблер vsyasm, Windows SDK 7.1. Всичко необходимо е описано в инструкциите включени в архивния файл, който може да се изтегли от [154]. Тази забележка е включена поради намерението да се публикува за свободно използване лежащия в основата на програмната система код. За целта е необходимо използваният код и програмни средства да са използваеми с достатъчно либерални лицензи.

"Работният материал" на пресмятанията с произволна точност: Малко за изчислимите реални числа и какво всъщност се моделира.

Някои предварителни факти.

Както е известно, реалните числа могат да бъдат аксиоматично въведени като единственото множество (с точност до изоморфизъм) със структура наредено поле, плюс допълнителна аксиома. Последната може да се формулира по различен начин: съществуване на точна горна граница на ограничено отгоре множество или аксиомата на Архимед и принципа на Кантор за вложените затворени интервали. Получената по този начин структура вече е пълно наредено поле. Освен това съществуват различни, но еквивалентни, методи за "построяване" на това множество, по такъв начин, че полученото множество да удовлетворява описаната по-горе аксиоматика. Един от тях е прототип на попълването на метрично пространство за случая, когато изходното метрично пространство е множеството на рационалните числа. Класовете от еквивалентни редици на Коши от рационални числа са новите елементи на попълненото множество. Аритметичните операции и наредбата са пренасят съвсем лесно в новопостроеното множество. В него вече всички редици на Коши имат граница, която принадлежи на това множество (реалните числа). Всяка формална система включваща възможност за описване на действия и записване на "ефективни процедури" (алгоритми) за пресмятане на реални числа използва, разбира се, някаква азбука Σ с краен брой символи (букви). Множеството Σ* на всички крайни думи (крайни поредици от букви), образувани от тези символи е изброима безкрайност. Оттук и невъзможността за именуване на всички реални числа (неизброима безкрайност, второ след това на естествените числа трансфинитно число, ако се приеме съответна допълнителна аксиома в теорията на множествата) с думи в Σ*. Това е следствие от построението, което включва безкрайни подмножества рационални числа (редици на Коши). Каквато и да е система за именуване на реалните числа с крайна азбука Σ по необходимост използва безкрайни думи в Σ^{ω}.

За "конструктивните" реални числа.

Интересно е, че дори да се ограничим до множеството на изчислимите (конструктивни) реални числа (те са изброима безкрайност - съответства на подмножество на Σ*, което е запис на алгоритмите за пресмятане, където Σ е азбуката на формалната система, включваща например символи за кодиране на числата в дадена позиционна система), проблемът с именуването остава, ако искаме да сме в състояние да извършваме ефективно операциите - в смисъл да сме в състояние да ги извършваме в краен брой стъпки за всяка предварително зададена точност. Това е така независимо от построението, в което влизат вече само изброимо множество редици: изчислими редици на Коши от рационални числа с изчислим регулатор на сходимост. Обобщението на обичайното определение на редица на Коши (или фундаментална редица) с регулатор на сходимост се налага, за да се може да се охарактеризира в термини на конструктивност и начина на сходимост, а не само факта, че такава има. За редица от рационални числа е $\{r_n\}$ функцията $h:\mathbb{Q}\to\mathbb{N}$ е регулатор на сходимост, ако за $\forall\varepsilon\in\mathbb{Q}^+$ имаме $n,m\geq h(\varepsilon)\Rightarrow|r_n-r_m|<\varepsilon$. Редицата е фундаментална, ако за нея съществува регулатор на сходимост. Ако $f:\mathbb{N}\to\mathbb{Q}$ е функцията задаваща дадена редица рационални числа с регулатор на сходимост *h,* то двойката (*f,h*) определя в общия случай реално число по обичайния начин и ролята на *h* е просто в съществуването му. Да предположим, че е зададена някаква изчислима номерация на рационалните числа $\nu:\mathbb{N}\to\mathbb{Q}$ (такива съществуват, дори натурална, т.е. $\mathrm{Dom}(\nu)=\mathbb{N}$). Редицата от рационални числа $\{r_n\}$ се нарича *изчислима*, ако за нея съществува общо рекурсивна функция $g:\mathbb{N}\to\mathbb{N}$, за която $r_n=\nu(g(n))$. Редицата от рационални числа $\{r_n\}$ се нарича *изчислимо сходяща*, ако за нея съществува изчислим регулатор на сходимост. Реалното число е изчислимо, ако е граница на редица рационални числа, която е едновременно изчислима и изчислимо сходяща [151]. По самото си определение множеството на тези редици е изброима безкрайност. Може да се докаже, че множеството на изчислимите реални числа ("конструктивен континуум", тук ще го отбелязваме с $\widehat{\mathbb{R}}$) е подполе на полето на реалните числа. Ефективни процедури с числа от така породеното множество обаче не могат да се извършват над съответното им кодиране като алгоритми за пресмятане (с подмножество от имена в Σ*, на което обаче липсва необходимата структура), а над имената, отразяващи структурата, и в самата си конструкция включващи безкрайни обекти. За сравнение, при неравенство на две рационални числа (двойки цели по построение) p/q < r/s е достатъчно да се провери неравенството ps < rq за цели числа, а аритметичните операции се свежда отново само до такива над цели числа. За (изчислимите) реални числа операцията се счита за ефективна, ако е възможна в краен брой стъпки за всяка предварително зададена точност. Точността може да се определя по различен (но в известен смисъл еквивалентен) начин в зависимост от използваната система за именуване. Такава може да бъде някаква позиционна система, в която точността се определя по дължината на префикса в безкрайното име, но може да бъде и редица от вложени затворени интервали с рационални граници, например. И накрая, не е маловажно да се отбележи, че разглеждания по-горе подход за въвеждането на конструктивни (изчислими) реални числа може да се преведе и на езика на абстрактно изчислително устройство (от тип машина на Тюринг). Това обаче изисква маса технически детайли, които биха отнели много място. Целта тук е по-скромна. Да се поставят в подходящ контекст пресмятанията с произволна точност на ниво идеи.

Възможни изводи.

От гледна точка на пресмятанията използваме само изчислими реални числа, тъй като други не могат да бъдат посочени по име (т.е. ефективно породени от някакъв алгоритъм). Тази позиция обаче се нуждае от едно уточнение. Включени са всички такива, които могат по принцип да бъдат изчислени (дори да не ни е известно в момента как), а не само тези, за които това вече е известно. Т.е. "съществуването" в някакъв по-широк смисъл, следващо от чисто логически допускания например, се нуждае от допълнително разглеждане: възможен ли е алгоритъм за представяне по принцип. Много поучителен е опитът за пренасяне на идеята, използвана в известното доказателство за това, че реалните числа в интервала [0,1] са неизброимо множество, в случая на изчислими реални числа. Ако допуснем, че съществува алгоритъм, който да поражда списък с изчислимите реални числа (представени в случая в някаква позиционна, да речем десетична система) в интервала [0,1], можем да построяваме поетапно число, различно от всички породени в този списък, опровергавайки факта, че са изброимо множество. Изводът е очевиден. Такъв алгоритъм не съществува. Това впрочем е следствие от теоремата на Райс (всяко нетривиално индексно множество, т. е. различно от $\varnothing$ и $\mathbb{N}$, е нерекурсивно; в дадения случай се имат предвид индексите на машините на Тюринг, пресмятащи съответно първите n знака в десетичното представяне на даденото число при вход n или пък номерата на съответните МНР - програми; МНР - машини с неограничени регистри). Тънкостта тук се състои в това, че множеството на изчислимите реални числа е изброимо, но в известен смисъл, "неконструктивно изброимо" [151]. Фактът, че мощността му може да бъде оценена като изброима безкрайност не дава автоматично и алгоритъм за номериране. Такъв в дадения случай просто не съществува. Примерът посочва с пределна яснота разликата в тълкуването на понятието "съществуване" в смисъл на непротиворечива възможност от една страна и реално посочване на алгоритъм за осъществяване, от друга. Това, че за дадено твърдение "съществува А", "не съществува А" е невярно, не означава "може да бъде ефективно посочено А". Горният пример е частен случай на по-общ резултат, който гласи, че каквато и да е номерацията му, $\widehat{\mathbb{R}}$ не е конструктивно изброимо спрямо нея (вж. [151] за определението на конструктивна изброимост). Много неща, разбира се, зависят от гледната точка, но за тези с по-конструктивно възприемане на математиката, твърдения от вида "всяко множество може да бъде добре наредено" (еквивалентно на аксиомата за избора) звучат почти като "теологични". Исторически обаче в математиката се считат за съществуващи тези обекти, които са непротиворечиви. Използването на неконструктивни доказателства с твърде силни средства от типа на аксиомата за избора (в най-силния ѝ, "неорязан" вид: "За всяко семейство X от непразни множества съществува функция f, която съпоставя на всяко едно множество от семейството един от елементите на това множество [150]. Функцията f се нарича функция на избора за даденото семейство.", формално $\forall X\left[\varnothing \notin X \Rightarrow \exists f : X \to \bigcup X, \forall A \in X\left(f(A) \in A\right)\right]$; съществуват варианти на аксиомата с ограничения, касаещи мощността на множествата) се смятат за нещо нормално. Редица "патологии" от тип съществуване на неизмеримо множество от реални числа не би съществувал, ако такива средства не се използват (не може да се построи пример за неизмеримо в смисъл на Лебег множество, който да не използва аксиомата за избора, показано е от Solovay [126]). Има обаче и резултати, например теоремата на Тихонов за това, че произведението на *произволен* брой компактни топологични пространства е компактно (еквивалентна на аксиомата за избора [84]), които се смятат за неотменна част от математиката. Терминът "теологични" не е пресилен или пренебрежителен. Болцано по повод на разглеждане на естествените числа в цялата им съвкупност казва, че той не ги

вижда всички едновременно, но Бог ги вижда. Кантор по повод цитат от библията - "Dominus regnabit in infinitum (aeternum) et ultra" (Бог властва в безкрайността (вечността) и отвъд), твърди: " Ich meine dieses 'et ultra' ist eine Andeutung dafür, dass es mit dem ω nicht sein Bewenden hat, sondern dass es auch darüber hinaus noch etwas giebt." (Мисля, че това "и отвъд" е намек, че и след ω има още нещо) [173]. По наше мнение просто в математиката има две нива на съществуване - в смисъл на непротиворечивост и в смисъл на конструктивна построимост. Освен това, както бе споменато, някои алгоритми очакват откриването си, така че не напразно има формулировка за относителна изчислимост ("алгоритъм с оракул") спрямо някаква система от функции. Малко вероятно е това да промени нещата по същество в понятието изчислимост, но някои неща, свързани с установени оценки за ефективност, могат и да претърпят промени във връзка с развитието на идеята за квантови компютри. Впрочем, проблемът със съществуването е по-дълбок и присъства дори на нивото на по-общото тълкуване на съществуването като логическа непротиворечивост. Някаква насока за осмисляне се дава от това, че във всяка достатъчно богата логическа теория (включваща поне аритметиката от първи ред, с квантори по променливите) може да се формулира твърдение, което е нито доказуемо нито опровержимо в рамките на теорията (Гьодел). Правени са изключително много тълкувания, но според нас това означава, че не може да има теория на всичко - всяка една теория е просто модел на част от реалността и изучаването на различните възможни следствия е занимание по-скоро с изучаването на специфична способност за моделиране. Фиксирането на езика (формалната система) носи своите ограничения. На интуитивно ниво част от възникващите проблеми са свързани с опит да се моделира непрекъснатост ("континуум") с дискретни средства - с други не разполагаме. Всяка формална система включва своята крайна азбука от символи [125]. Това е *фундаментално* ограничение, което не можем да избегнем. Дори Кантор при неформалното описание на множество говори за "завършен акт на мисълта" в смисъл на дискретно състояние. Друга част от проблемите са свързани с наредбата, като неизменна част в арсенала от средства за моделиране. Тук нещата според нас отново са свързани със собствената ни способност за отразяване (моделиране). Съществото тук е, че не можем да преведем по друг начин представата си за реалността освен чрез причинно-следствена връзка и то *осъществявана на локално ниво*. Причината предшества следствието, пораждайки наредба. Някаква параметризация на състояния, които бихме могли да възпроизведем *винаги* в рамките на моделирането. Без тази парадигма за *безусловна възпроизводимост на явленията* съвременната наука не може да съществува. Дали това е единственият път за "разбиране" не е ясно, но е сигурно, че така се получава "обяснение". Впрочем, като оставим настрана философските виждания на Кант (по-късно и на Гьодел; времето не е произтича от физическата реалност, а е "ефект на наблюдателя" - схващайки причинно-следствената зависимост като линейна наредба - няма никакво доказателство за това) за субективната същност на времето, обобщения в тълкуването на времето като единствена естествената параметризация съществуват [в рамките на ОТО или ограничението в точността на измерване - времето не е независима променлива от енергията и за дадена частица (в някакъв смисъл "Фурие - допълнителни"; едната в "света на честотите", другата, в "света на наблюдателя - Фурие преобразуваните честоти") $\Delta t \Delta E = h / 2\pi$ - в квантовата механика]. Малко известен факт е обаче, че класическата механика може да бъде построена по безвремеви начин - в смисъл, че времето не е превилигирован параметър (естествения параметър на локалния наблюдател) - а просто като съотношение на променливи (част IV в [118]). В заключение, без да се избира конструктивизмът като единствено заслужаващ внимание, това е най-близката позиция, на която може да се застане, когато става дума за реално *практическо* решаване

на каквито и да е модели, възникнали в различни области. Оттам нататък акцентът вече е върху начина на намиране и ефективността за получаване на резултата. Произходът на самия модел с неговата ограниченост до способностите да се отразява реалността бива забравен.

Помощни средства за изграждане на програмни инструменти.

Удобното използване на програмни инструменти, базирани на инструменталните средства в основните библиотеки с функции и математически методи, има известни изисквания към интерфейса. Едно такова изискване е потребителят да е в състояние да формулира задачата си по естествен начин, без да пише допълнителен програмен код свързан с всяка отделна задача. В противен случай би се изисквала повторна компилация, т.е. допълнително изискване и за наличие на софтуер за разработка, който няма пряка връзка с решаваната задача от потребителска гледна точка. Това в много от случаите води до необходимостта от инструменти, осигуряващи прозрачен превод на въведен от потребителя текст в термините на някаква математически израз, обикновено интерпретиран като функция и евентуално нейни производни, които вече са реалният вход към програмата, решаваща даден клас задачи. Подобен спомагателен интерфейс е полезен и на етапа на разработката на конкретен програмен инструмент, спестявайки време, например при многобройни тестове. Помощните средства за съставяне на такъв интерфейс в настоящата програмна система са две: интерпретатор на математически изрази и символно диференциране.

Разбира се, използването на интерпретатор е оправдано само при проектиране на програмен инструмент, който е предназначен за решаване на *самостоятелни* задачи от даден клас, при които бързината на изпълнение не е от критично значение. Все пак интерпретаторът води до известно забавяне. В настоящата програмна система пример за това е намирането на корен на скаларно нелинейно уравнение. Необходимата програма може да бъде надстроена с интерпретатор до самостоятелен програмен инструмент (например разработените от нас MPRootFindTestLocal или MPRootFindTestNH), но в случая, в който се използва като подзадача (например за получаване на някои математически константи от типа константа на Рамануджан-Солднер), входът към решателя (включващ функция и производна) е кодиран непосредствено в извикващата програма.

Дотук се обсъждаха помощните средства, касаещи интерпретацията на входните данни. Друг аспект на усъвършенствания интерфейс е интерпретацията на изходните данни. Това се отнася за случая, когато полученото решение не е просто число, а поредица числа, които могат да се представят графично. Например при численото решаване на (системи) обикновени диференциални уравнения, решението (или поне проекцията му върху някаква равнина на част от променливите) може да се представи като крива. Дори представянето на обикновена функция графично има редица предимства при решаване (или по-точно преди решаването) на някои задачи. Например преди намирането на корен на уравнение, графичното представяне на функцията дава груба представа за локализирането на съответния корен, както и качественото поведение на функцията в околност на корена. Това е изключително полезно, ако ще се прилага някакъв метод с локална сходимост. В настоящата програмна система визуализацията се използва по два

начина. Визуализация на вече изцяло генерирани данни за решението и динамична визуализация в процеса на решаване (кривата се разгъва постепенно). И в двата случая се използва сходен програмен подход, състоящ се в извикване на програмата gnuplot от дадения програмен инструмент, с вход генерираните от него данни. Това са и единствените места в програмната система, които разчитат на готова, външна за системата програма (безплатна). Беше преценено, че самостоятелна визуализираща програма е възможна, но разработката ѝ би била излишна в момента при наличие на работещо решение. Впрочем самостоятелната за системата визуализация не е отречена възможност, но е отложена за момента (както и редица други функционалности, всяка от които се коментира на съответното място). Описанието на схемата за визуализация чрез gnuplot е дадено по-надолу.

Интерпретатор на математически изрази.

Описанието е дадено накратко малко по-долу, а тук са изброени само разпознаваните от парсера на интерпретатора на системата функции и оператори.

0-местни функции (константи): e, pi, gamma, G.

Последните две са съответно константата на Ойлер-Маскерони $\gamma = \lim_{n\to\infty}\left(\sum_{k=1}^{n}\frac{1}{k} - \ln(n)\right)$ и константата на Каталан $G = \beta(2) = \sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)^2}$.

1-местни функции: exp, ln, sqrt, sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, asinh, asosh, atanh, Ai, Bi, S, C, Si, Ci, erf, erfc, EllipticK, EllipticE, Gamma, Zeta, abs, Ei, li.

Тук Ai, Bi са функциите на Ейри. S, C са интегралите на Френел. EllipticK, EllipticE са съответните пълни елиптични интеграли. Gamma и Zeta са едноименните функции на Ойлер и Риман (обикновено означавани с Γ и ς, Γ-функцията е известна и като интеграл на Ойлер от втори род). Ei, li са експоненциален и логаритмичен интеграл съответно.

2-местни функции: Beta, GammaHigh, GammaLow, BesselJ, BesselY, BesselI, BesselK, JacobiSN, JacobiCN, JacobiDN, JacobiNS, JacobiNC, JacobiND, JacobiSC, JacobiCS, JacobiSD, JacobiDS, JacobiCD, JacobiDC, PowInt.

Beta е така наречения интеграл на Ойлер от първи род. Следват двете непълни гама функции. После са съответните функции на Бесел и на Якоби. PowInt осъществява повдигане на цяла степен.

1-местни оператори: "-","+"

2-местни оператори: "+", "-", "*", "/", "^"

Последният е повдигане на степен. Операторът минус се различава естествено автоматично от контекста.

Операторът за повдигане на степен е реализиран в пълна общност и минава през пресмятане на логаритъм и експонента. В контекста на интерпретатора данните са винаги числа с плаваща запетая, така че за осъществяване на много по-бързо повдигане на цяла степен е добавена за удобство функцията PowInt.

Процесът на пресмятане на текстово записан математически израз, включващ алгебрични операции и функции, протича на три етапа. Първият етап (tokenization) е разделяне на текстовия низ на отделни абстрактни символи (tokens), представящи

самостоятелни единици на пресмятането: данни и оператори. Данните могат да са буквално записани числови константи, символни константи или символни параметри. В последния случай параметърът се замества чрез отделна интерактивна процедура с числова константа преди пресмятането. Операторите са алгебрични и функционални. На втория етап полученият в инфиксна форма абстрактен израз се преобразува с постфиксна форма (обратен полски запис), в който вече липсват разделители (алгебрични и функционални, в конкретния случай са скоби и запетаи). Третият етап е същинското пресмятане на израза, използващо обратния полски запис.

Следва кратко описание на етапите в обратен ред.

След реализирането на класическата схема за преобразуване на израз (включващ аритметични и функционални оператори) в обратен полски запис, интерпретацията е елементарна (преобразуваният израз е поставен в някаква подредена последователност). В този случай се използва абстрактният израз, който е резултат от работата на втория етап. Използва се стек за междинните резултати (работен стек, стек на интерпретатора): Изтегля се поредният символ. Ако символът (token) представя данни (число с текущата за пресмятането точност), те се поставят (операция push) в стека на интерпретатора. Ако символът е едноместен (унарен) оператор, то от стека на интерпретатора се извлича (pop) неговият операнд и се извършва операцията, след което получените данни се записват (push) обратно в работния стек. Ако символът е на двуместен (бинарен) оператор, от стека на интерпретатора се извличат последователно двата му операнда, извършва се операцията и резултатът се записва обратно в стека на интерпретатора (тук има малка тънкост, трябва да се отчете дали операцията е ляво- или дясно-асоциативна). За повече аргументи (при функционален оператор) се действа аналогично. Процесът продължава до изчерпване на символите във входната последователност от символи на обратния полски запис на израза. Последната изчислена стойност е стойността на израза (или числото на върха на стека, ако няма никакви операции повече).

По-сложен е алгоритъмът за преобразуване на израз от обичайната инфиксна форма в постфиксна (обратен полски запис). Алгоритъмът използва входна последователност от символи (tokens) и стек. Резултатът се записва в изходна последователност. Схемата е следната:

Докато има още символи за четене

Ако символът е число, той се добавя към изходната последователност.

Ако символът е функционален оператор, той се поставя в стека.

Ако символът е разделител на аргументи на функция (в случая е запетая):

Докато символът на върха на стека не стане лява скоба, извличат се операторите от стека и се поставят в изходната последователност.

Ако в стека не се срещне лява (отваряща) скоба, то или е пропуснат разделител (запетая) или има липсваща лява скоба.

Ако символът е оператор op1, то:

Докато на върха на стека има оператор op2 и или операторът op1 е ляво-асоциативен и приоритетът му е по-малък или равен на този

на op2, или операторът op1 е дясно-асоциативен и приоритетът му е по-малък от този на op2:

op2 се извлича от стека и се поставя в изходната последователност.

op1 се поставя в стека.

Ако символът е лява скоба, поставя се в стека.

Ако символът е дясна (затваряща) скоба:

Докато на върха на стека не се появи лява скоба, операторите от върха на стека се преместват в изходната последователност.

Лявата скоба се отхвърля (изтегля се без да се добавя в изходната последователност).

Ако символът на върха на стека е функционален, да се добави в изходната последователност.

Ако стекът се изпразни преди да се срещне символ за отваряща скоба, то в израза е пропусната скоба.

Ако вече няма входни символи

Докато в стека има още символи

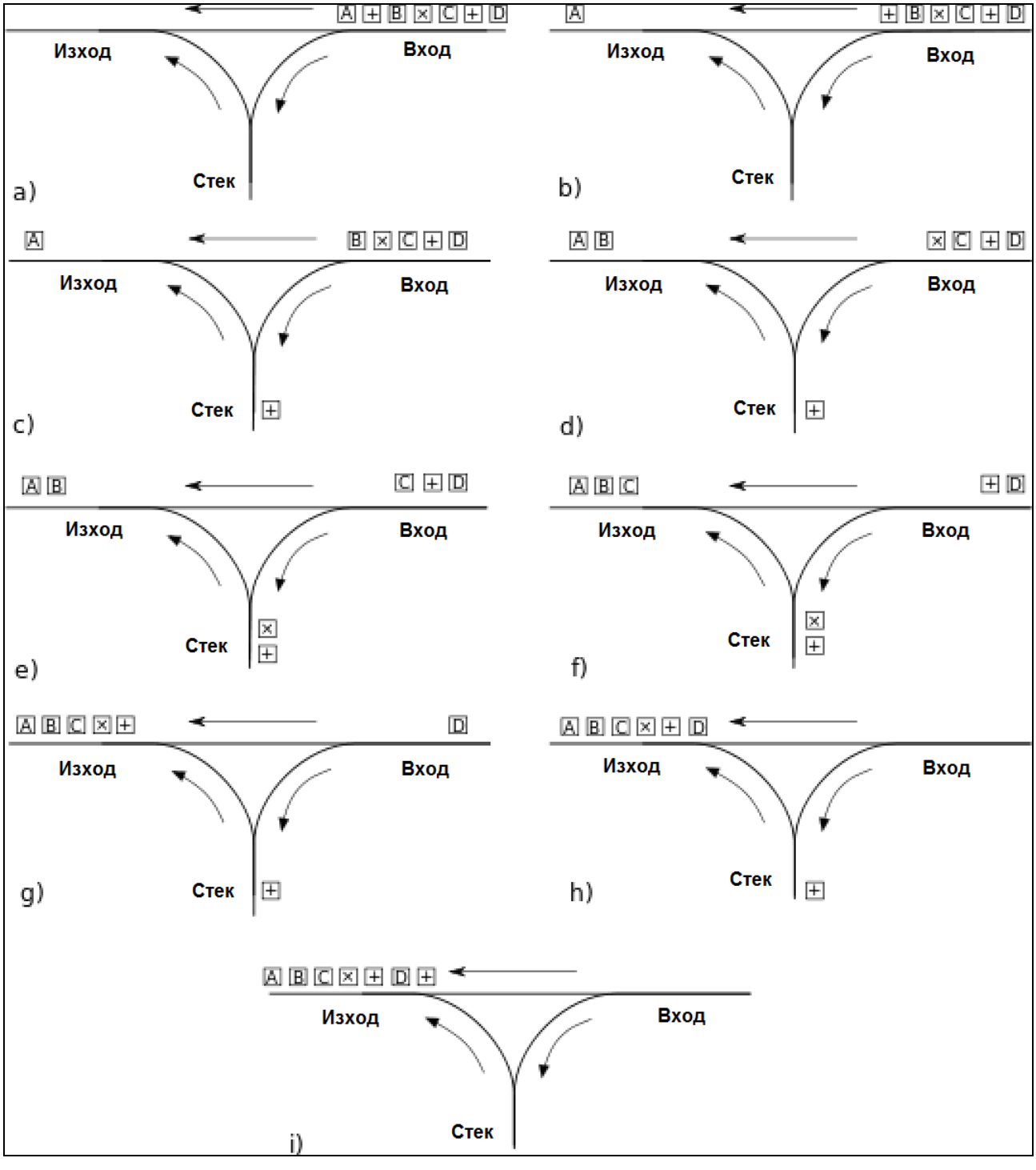
Ако символът на върха на стека е отваряща скоба, то в израза има незатворена скоба.

Преместват се последователно всички оператори от върха на стека в изходната последователност.

Край.

Фигура 1. Алгоритъм за преобразуване в обратен полски запис.

По-долу има илюстративна картинка на процеса за простия израз $A + B \times C + D$, който се преобразува в $ABC \times + D +$.



Фигура 2. Пример за преобразуване в обратен полски запис.

Процесът на разделяне на първоначалния текстов низ на абстрактни символи е прост като идея, но изисква известно внимание при реализация. В даден момент на преобразуването се прочита следващ символ. В зависимост от текстовия символ в текущата позиция са възможни няколко варианта на символ (абстрактен): лява скоба, дясна скоба, запетая, алгебричен оператор или идентификатор. В последния случай се

проверява дали е функционален оператор, символна константа или параметър. Съществената част е всъщност в проверката за валидност на поредния извлечен символ. За целта се използва еквивалент на краен автомат, който мени състоянието си след всеки прочетен символ и решава дали следващият символ в потока е допустим (не може например да има идентификатор след дясна скоба, запетая след алгебричен оператор и т. н.). За целта с използват логически променливи, които показват дали в даденото състояние е допустим съответен символ. Ето част от кода, който дава представа за какво става дума:

```
CanBeVar = (isArithmeticOperator(tk) || isLP(tk) || isSeparator(tk));
CanBeRConst = (isArithmeticOperator(tk) || isLP(tk) || isSeparator(tk));
CanBeMConst = (isArithmeticOperator(tk) || isLP(tk) || isSeparator(tk));
CanBeFunc = (isArithmeticOperator(tk) || isLP(tk) || isSeparator(tk));
CanBeFunc2 = (isArithmeticOperator(tk) || isLP(tk) || isSeparator(tk));
CanBeArOp = (isVariable(tk) || isMathConst(tk) || isRealConst(tk) || isLP(tk) ||
isRP(tk) || isSeparator(tk));
CanBeLP = (isFunction(tk) || isFunction2(tk) || isArithmeticOperator(tk) || isLP(tk) ||
        isSeparator(tk));
CanBeRP = (isVariable(tk) || isRealConst(tk) || isMathConst(tk) || isRP(tk));
CanBeSep = (isVariable(tk) || isRealConst(tk) || isMathConst(tk) || isRP(tk));
CanBeUnary = (isLP(tk) || isSeparator(tk) || pos == 0);
```

Програмно в самия символ (на етапа на създаването му) се запазват полета за тип, текстово представяне, приоритет, брой аргументи, вид асоциативност, стойност и флаг за това дали е данни (т.е. кои от предишните полета всъщност касаят конкретния символ).

Преди същинското пресмятане (след попълване на всички символни константи и параметри), символите в обратния полски запис се опростяват. Съдържат единствено поле за кода на операция и поле за данни.

Етапите преди същинското пресмятане са в отделен файл mpir_parse_fx.cs. Той съдържа необходимите структури от данни за абстрактните символи на израза (основен и опростен, вж. програмния фрагмент по-долу),

```
public class Token
{
    private string _type;
    private string _repr;
    private int _prcd;
    private int _args;
    private string _assoc;
    private mpir.mpf_t_val;
    public bool data;

    public Token()
```

```
    {
        this._type = "UNKNOWN";
        this._repr = "";
        this._prcd = -1;
        this._args = -1;
        this._assoc = "L";
        this._val = mpir.mpf_init_set_ui(0);
        this.data = true;
    }
    public Token(string type, string repr)
    {
        this._type = type;
        this._repr = repr;
        this._prcd = -1;
        this._args = -1;
        this._assoc = "L";
        this._val = mpir.mpf_init_set_ui(0);
        this.data = true;
    }
    public string type
    {
        get { return this._type; }
        set
        {
            if (value.GetType() == this._type.GetType())
            {
                this._type = value;
            }
            else
            {
                throw new ArgumentException("Invalid Type");
            }
        }
    }
    // .........
    // Останалите свойства
    // .........
}

public struct SToken // simplified token
{
    public int opcode;
    internal mpir.mpf_t val;
    // public string repr;
}
```

Фигура 3. Структура на абстрактните символи извличани от първоначалния текстов низ, съдържащ математически израз.

както и клас ExpressionParser съдържащ двете основни функции, ExpToTok (играе ролята на парсер) със сигнатура

public List<Token> ExpToTok(string exp0, out int count0, out bool ok),

преобразуваща входящия низ, представящ израза, в списък от елементи (абстрактни символи) и ToPostfix със сигнатура

public List<Token> ToPostfix(List<Token> ltk, out List<string> vars, out List<string> mconsts),

получаваща на входа списъка получен от парсера и връщаща списък с елементите (абстрактните символи) в обратен полски запис, където vars и mconsts са списъците с променливи и константи.

Символно диференциране.

Традиционният начин за символно диференциране е свързан с построяване на дървовидна структура за математическия израз, върху която се прилага съответен алгоритъм. Предложен е нетрадиционен подход, който се основава на непосредствената обработка на вече получения в обратен полски запис израз, чиято обработка дава производната отново във вид на обратен полски запис. В процеса на разработката се оказа, че подобен подход вече е опитван [86]. Тази работа се оказа изключително полезна, например при специфичната обработка на съответните масиви с стил C (естествено, съответните части от програмата да се маркират като unsafe) - заради смисловата натовареност на използваните индекси като указатели (върху които се извършва съответната "аритметика на указателите" с цел посочване на конкретни подлежащи на промяна части от масива в зависимост от контекста). Има разбира се и допълнителни усложнения с 2-местните функции и опростяването на получения в обратен полски запис израз на производната, но като цяло, този подход работи и според нас заслужава внимание.

Идеята е проста, макар реализацията да не е съвсем тривиална. Построява се рекурсивна функция derive(int low, int upp, string dv), която в началото се прилага върху целия израз, а в процеса на обработка индексите low и upp, сочат подлежащата на обработка част (dv съдържа символа, по който се диференцира). Поддържат списъци pfxlist и derivat с елементите в обратен полски запис на породения с ExpToTok и ToPostfix първоначален израз и породената производна. Въвежда се и функция grasp, връщаща броя на операндите, които са в обхвата на оператора с текущия индекс, както и няколко помощни функции и масиви с цел точното установяване на долната и горна граница за следващото рекурсивно извикване. Освен това предварително се построяват специални елементи (tokens), които се използват в процеса на пораждане на производната (Token one, tkmul, tkbessel и т.н.). Функцията е доста дълга поради различните случаи за обработка, но следващата част от кода е достатъчно представителна:

```
unsafe public void derive(int low, int upp, string dv)
{
   Token last = pfxlist[upp];
   int count, k;

   if (last.args == 0)
   {
      if (pfxlist[upp].repr.Equals(dv)) derivat.Add(one);
      else derivat.Add(tk0); // add 0
   }
   .............
   else if ((last.args == 2) && last.repr.Equals("*"))
   {
      if (low < 0 || upp < 1 || upp - 2 - grasp_arr[upp - 1] < 0) return;
      else
      {
         stidx1 = low;
         for (k = low; k <= upp - 2 - grasp_arr[upp - 1]; k++)
         {
            derivat.Add(pfxlist[k]);
         }
         stidx2 = derivat.Count;
         derive(upp - 1 - grasp_arr[upp - 1], upp - 1, dv);
         derivat.Add(tkmul);
         stidx1 = derivat.Count;
         derive(low, upp - 2 - grasp_arr[upp - 1], dv);
         stidx2 = derivat.Count;
         for (k = upp - 1 - grasp_arr[upp - 1]; k <= upp - 1; k++)
         {
            derivat.Add(pfxlist[k]);
         }
         derivat.Add(tkmul);
         derivat.Add(tkplus);
      }
   }
   .............
   else if (last.args == 2) // binary function
   {
      switch (last.repr)
      {
      .............
         case "BesselJ":
            bool haspow = true;
            if (haspow)
            {
               for (k = low; k <= upp - 2 - grasp_arr[upp - 1]; k++)
               {
```

```
                derivat.Add(pfxlist[k]);
            }
            for (k = upp - 1 - grasp_arr[upp - 1]; k <= upp - 1; k++)
            {
                derivat.Add(pfxlist[k]);
            }
            derivat.Add(tkpowop);
        }
        for (k = low; k <= upp - 2 - grasp_arr[upp - 1]; k++)
        {
            derivat.Add(pfxlist[k]);
        }
        for (k = upp - 1 - grasp_arr[upp - 1]; k <= upp - 1; k++)
        {
            derivat.Add(pfxlist[k]);
        }
        derivat.Add(one);
        derivat.Add(tkminus);
        derivat.Add(tkbesselj);
        if (haspow) derivat.Add(tkmul);
        for (k = low; k <= upp - 2 - grasp_arr[upp - 1]; k++)
        {
            derivat.Add(pfxlist[k]);
        }
        for (k = upp - 1 - grasp_arr[upp - 1]; k <= upp - 1; k++)
        {
            derivat.Add(pfxlist[k]);
        }
        derivat.Add(tkbesselj);
        derivat.Add(tkminus);
        for (k = low; k <= upp - 2 - grasp_arr[upp - 1]; k++)
        {
            derivat.Add(pfxlist[k]);
        }
        derivat.Add(tkdiv);
        derive(low, upp - 2 - grasp_arr[upp - 1], dv);
        derivat.Add(tkmul);
        break;
      default:
        MessageBox.Show("Sorry, no authomatic derivative for " +
last.repr);
        break;
    }
  }
  .............
  else if (last.args == 1) // unary function
  {
    switch (last.repr)
```

```
      {
         case "exp":
            for (k = low; k <= upp - 1; k++)
            {
               derivat.Add(pfxlist[k]);
            }
            derivat.Add(tkexp); // derivat.Add(last);
            derive(low, upp - 1, dv);
            if (derivat[derivat.Count - 1].repr.Equals("1"))
            {
               derivat.RemoveAt(derivat.Count - 1);
            }
            else
            {
               derivat.Add(tkmul);
            }
            break;
         .............
      }
   }
   .............
}
```

Фигура 4. Част от кода на функцията, осъществяваща символно диференциране без използване на дървовидна структура.

Заключителната част са процедури на опростяване, премахващи части от типа x + 0, 1*x, (x + y)*0 и т.н., които няма да привеждаме.

Автоматична визуализация с помощта на gnuplot.

За използването на gnuplot като спомагателна визуализираща програма, извиквана от друг програмен инструмент, в него трябва да инициализира нов процес. Ако предположим, че в работната директория на този програмен инструмент има текстов файл gnuplot.txt, чието съдържание е адресът (пътят във файловата система) на програмата gnuplot, то това се постига например с

```
try
{
   StreamReader gnuplotpath = new StreamReader("pgnuplot.txt");
   path = gnuplotpath.ReadLine();
   gnuplotpath.Close();
}
catch (IOException)
{
   Console.WriteLine("Invalid location or missing file 'pgnuplot.txt'");
   return;
}
```

```
gnuplot = new Process();
gnuplot.StartInfo.FileName = path;
gnuplot.StartInfo.UseShellExecute = false;
gnuplot.StartInfo.RedirectStandardInput = true;
gnuplot.Start();
```

Ключовият момент е в пренасочване на стандартния вход на gnuplot към поток за писане, в който програмният инструмент пише:

```
StreamWriter gnupStWr = gnuplot.StandardInput;
```

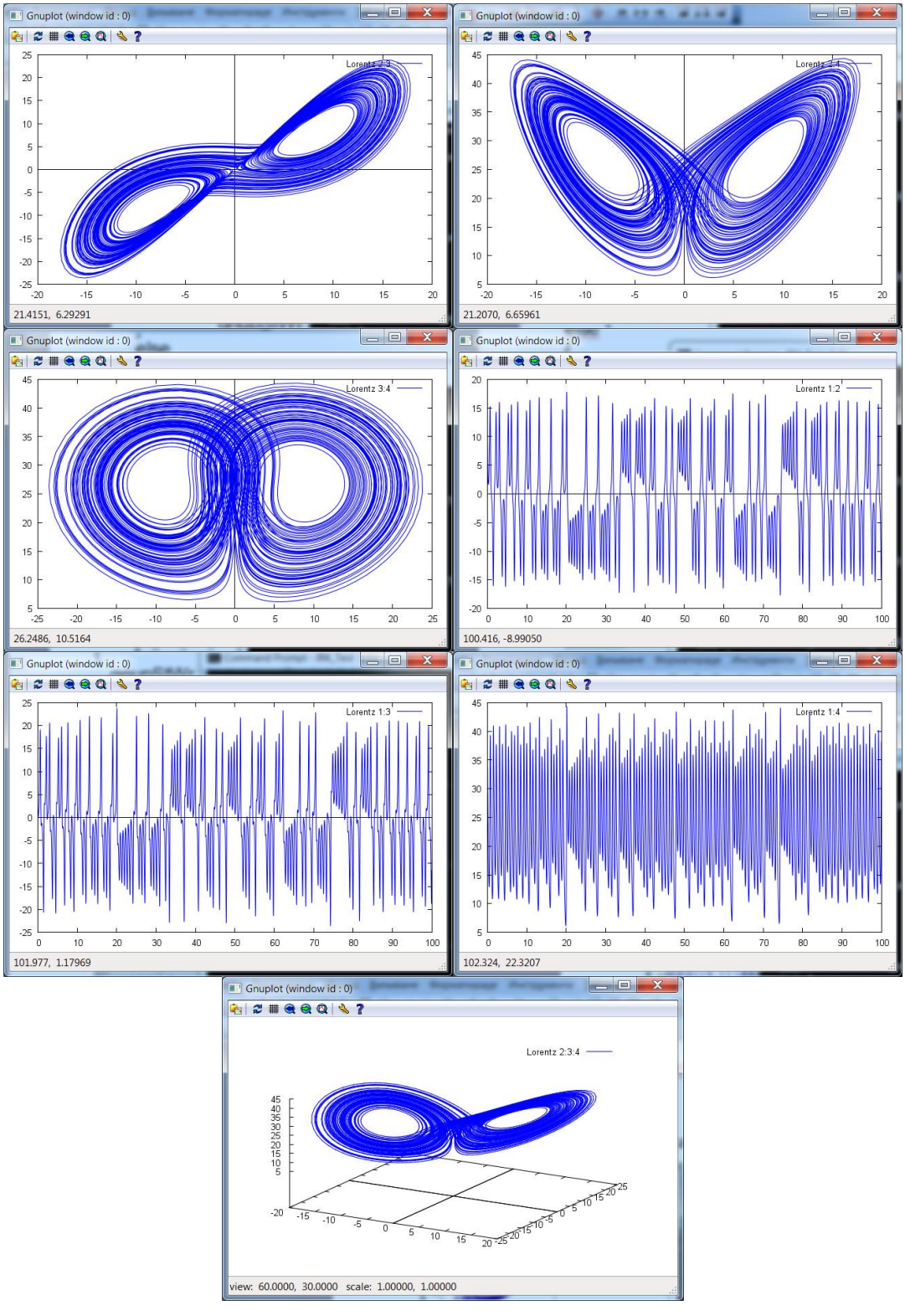
Оттам нататък той може да извежда в gnuplot както желаните данни, така и информация, управляваща визуализацията. Нека примерно има файл irktest.dat, съдържащ резултатите от пресмятане на система от 3 ОДУ от първи ред - в случая системата на Лоренц с подходящо избрана начална точка (в близост до атрактора) - по четири записа във всеки ред. Първият за променливата на времето, останалите за променливите x, y, z.

Програмният фрагмент по-долу (на title е присвоена преди това стойност "Lorentz")

```
if (mode == 0)
{
   ax1++; ax2++;
   plane = ax1.ToString() + ":" + ax2.ToString();
}      }
else if (mode == 1) plane = "2:3:4";
.....
title = title + " " + plane;

gnupStWr.WriteLine("set style data lines");
gnupStWr.WriteLine("set zeroaxis ls 7");
if (mode == 0)
{
   gnupStWr.WriteLine("plot 'irktest.dat'" + " using " + plane + " with lines lc
rgbcolor \"blue\" title" + title);
}
else
{
  gnupStWr.WriteLine("splot 'irktest.dat'" + " using " + plane + " with lines lc
rgbcolor \"blue\" title" + title);
}
```

извежда различни картинки в зависимост от формирането на параметрите ax1, ax2 и mode.



Фигура 5. Програмно задаване на различни варианти за 2D и 3D визуализация с извличане на данни от един и същ файл.

Тук умишлено са опростени нещата с възможната параметризация при mode = 1, за да не се налага да показваме всички 3D комбинации.

Случаят с динамично генериране на визуализация не се различава принципно. Просто програмното решение тогава не изглежда толкова елегантно. Използва се временен файл за решението, в който програмният инструмент пише, а gnuplot визуализира. Визуализацията повтаря и вече стари данни. Може, разбира се, визуализацията да не се прави на всяка стъпка, а през определен интервал от добавени към решението точки.

Предварителен бележки относно засегнатите теми по глави.

В този раздел са обсъдени идеи за мотивировката, контекста, методологията и по-общи перспективи, свързани със съответните теми, за да не прекъсва изложението в следващите глави, имащо по-конкретен характер и касаещо по-скоро въпроса "как се пресмята" (изключение прави уводният раздел в глава 3).

Към глава 1. Методи и средства за подобряване на пресмятанията на елементарни и специални функции с висока точност.

Пресмятането на елементарните и някои специални функции е ядрото на всяка система за математически пресмятания с висока точност. В много случаи те влизат в самото описание на конкретната задача, която предстои да се решава. Тук, разбира се, се включват и някои често използвани математически константи, използвани често в пресмятането на тези функции. От програмна гледна точка, това са едноместни функции, чийто аргумент е изискваната точност, извиквани в процеса на пресмятанията. Такъв подход позволява да се използват в различен контекст на изисквана точност, което наистина се случва. Възможен е малко по-различен подход в случая, когато предварителният контекст е известен и желаните константи се записват в глобални статични полета с еднократно извикване на съответните функции. В този случай реално пресмятане се прави само ако текущо изискваната точност е по-голяма от тази, с която е била пресметната константата в съответното статично поле. Този подход е използван в специалната, последно създадена част на програмната система, демонстрираща възможност за пресмятане на съвременните многоядрени процесори за паралелни пресмятания (като пример може да служи пресмятането на някои математически константи в глава 5). Там съответната процедура евентуално инициализира предварително глобални за класа полета, например за π и/или ln(2), ако е необходимо. Впрочем, паралелни пресмятания са възможни и за някои елементарни и специални функции, но към този момент тази възможност не е реализирана. Както беше споменато вече, начинът на разпаралеляване на пресмятанията е от решаващо значение. В зависимост от нивото, на което се прави, паралелното пресмятане не е необходимо винаги и за всичко. Въпреки това, осъвременяване на тази част от програмната система (която е и най-старата, въпреки че от време на време някои неща се заменят с малко по-добри) е една от бъдещите задачи по разширяване на системата - по-конкретно, създаване на алтернативи с паралелни пресмятания, където е възможно, с даване на възможност за разпаралеляване и на ниско ниво. Като следствие, това ще доведе вероятно до втори вариант на основната библиотека, ако държим да запазим имената и сигнатурите на сегашните функции. Пренасянето на вече готови резултати, описани в глава 5 "Бързо пресмятане на математически константи с висока точност " и евентуални промени в пресмятанията на някои елементарни и специални функции в такава втора,

алтернативна библиотека, не е изцяло завършено. Трябва да се отбележи, че в настоящата реализация на библиотека с математически функции често е давано предимство на методите ползващи бързо сходящи редове навсякъде, където това е възможно и практично (т.е. все пак достатъчно ефективно). Почти винаги съществуват алтернативни методи (непрекъснати дроби, интегрални представяния, използване на итеративни отношения и пр.), но за пресмятане с произволна точност е важна и възможността за проста оценка на грешката. Възприетият подход не е лишен от свои специфични изисквания, например при осигуряване на съгласуваност на основния ред и асимптотичното представяне (за спецификата на асимптотичните представяния с разходящи редове вж. [105]). Въпреки това, този подход е като че ли по-прост в контекста на произволна точност, която е все пак предварително задавана. В тази глава е предвидено описание на използваните методи за пресмятане. В значителна степен това описание носи и справочен характер. По-подробно описание има на местата, в които се използва нещо, които се нуждае от обяснение или има алтернативи, реализирани вече по различни начини. Основните ползвани източници са [106], [2], [30], [36], [34], [25], [70], [105], [69], както и многобройни сайтове, едни от най-полезните от които са [145], [143], [104]. Допълнителни препратки са дадени на място. Освен елементарните и специални функции в основната библиотека (файл funcs.cs) са включени и някои методи за други пресмятания. От тях в тази част ще бъдат разгледани само на някои (например числата на Бернули), а останалите ще бъдат описани по-нататък. В библиотеката много от функциите се използват вътрешно за изразяване на други (в съответен диапазон или за тип индекс например) и някои от тях (тета функции, числата на Бернули) не са специално представени в програмата SFCALC, описана в последния раздел на главата. Типичен пример са частните случаи на хипергеометричната функция

$$ {}_pF_q\left(a_1,\cdots,a_p;b_1,\ldots,b_q;z\right) \;=\; \sum_{n=0}^{\infty}\frac{\left(a_1\right)_n\ldots\left(a_p\right)_n}{\left(b_1\right)_n\ldots\left(b_q\right)_n}\frac{z^n}{n!}. $$

Използват се само частните случаи p = 0, q = 1 и p = 1, q = 1, при които (p < q + 1) редът е сходящ навсякъде (ако b = b_1 не е отрицателно цяло число естествено) и то за част от диапазона на променливата. При големи стойности на аргумента се използват други представяния.

Към глава 2. Методи и средства за подобряване численото пресмятане на някои класове определени интеграли.

Първо да се уточни, че става дума за еднократни интеграли. Основната цел за създаване на специални програмни инструменти за пресмятането им с много висока точност е свързана с възможното използване на резултатите в процедура от тип идентификация на константа. Задачата, разбира се, е важна и сама по-себе си. Като стар клон на числения анализ, темата е много добре изследвана и с многобройни подходи, засягащи както общата теория, така и конкретни класове функции. Не се цели обаче да се прави описание и анализ на всички възможни подходи. Един съвременен обзор, включващ и по-съвременни методи е [147]. Можем да споменем, че за интегриране с квадратурни формули от гаусов тип в системата всичко е извършено, с изключение на самостоятелен програмен инструмент. На практика такъв тип формула се използва вътрешно при реализация на неявна схема на Рунге-Кута при решаване на системи ОДУ, за която в системата ни има реализация (вж. в главата за

намиране на корени на скаларни нелинейни уравнения мястото, в което става дума за тестовете, свързани с намиране на корени на класически ортогонални полиноми).

И сега някои забележки за относителните достойнства на различните типове квадратурни схеми. От практическа гледна точка задачата за намиране на числената стойност на даден определен интеграл с предварително зададена точност почти никога не се свежда до еднократно прилагане на конкретна квадратурна формула. Вероятно най-голямото предимство за съответната квадратурна схема е възможността за повторно използване на данни от вече направени пресмятания. В този смисъл, квадратурни схеми от тип tanh-sinh (вариант на метода на двойната експонента - Double Exponential Transformation) или Clenshaw-Curtis имат известно предимство пред квадтатурните схеми използващи квадратура от гаусов тип. Първите могат да използват пресметнатите вече абсциси и стойностите на подинтегралната функция в тях и на следващи нива. При tahh-sinh квадратурата, могат да се използват и вече пресметнатите преди това тегла, но това не е от решаващо значение. При квадратура от гаусов тип правилата от тип Гаус-Кронрод са стъпка в тази посока, но все пак, ако се налага неколкократно пресмятане с все по-голяма точност, това не е достатъчно. В общия случай за различни конкретни задачи, най-подходящият тип квадратурна схема е различен. Връщайки се на квадратурните формули от гаусов тип, в случая, когато многократно пресмятане за достигане на определена точност не се изисква от контекста на решаваната задача, те са много полезни, а в някои случаи и необходими. Такъв е споменатият по-горе случай на вътрешното им използване при интегриране на диференциални уравнения с неявна схема на Рунге-Кута.

Името двойно експоненциална формула (или формула на двойната експонента) е дадено по очевидна причина – начинът, по който производната на преобразованието $g'(t)$, $g(t) = \tanh\left(\frac{\pi}{2}\sinh(t)\right)$, намалява в безкрайност. Това е най-добрата схема за числено интегриране на функции с особености в краищата на интервала при изискване за висока точност (около 1000 знака и повече) [12]. Квадратурните формули на Clenshaw-Curtis и тези от гаусов тип също заслужават внимание, но в случая, когато подинтегралната функцията има достатъчно добро поведение в целия затворен интервал. Разбира се, за да е гарантирано добро поведение на схемата tanh-sinh, функцията трябва да изпълнява определени изисквания. В [136] са приведени точни математически резултати за класове от функции, към които са приложими различни видове квадратурни формули от тип двойна експонента. Оригиналният източник е [135]. Много обширен обзор на историята на откриването на двойно експоненциалната формула, както и последващото развитие и приложения на идеите, свързани с нея в различни области, може да се намери в [100]. Простата евристична обосновка на идеята посредством формулата на Ойлер-Маклорен е от сравнително скоро (вж. например [19]). Учудващо е, че плеяда знаменити математици от миналото, които са посветили време и усилия в областта на числените квадратури, не са се натъкнали на тази идея. Нещо повече, минават доста години от началното публикуване на метода, преди той да бъде забелязан и оценен чрез по-широко внедряване в различни пакети и среди за числен анализ, които включват числени квадратури.

Квадратурната формула на Кленшоу-Къртис е малко по-стара [155]. Тя може да се изведе по различни начини. Свързана е по естествен начин с разлагането по полиноми на Чебишев и всъщност абсцисите на квадратурната формула са именно корените на полиномите на Чебишев, но ние няма да изследваме тази връзка (вж. [156] и [157]). Евристичната обосновка на изключителната ѝ ефективност за някои класове

подинтегрални функции е свързана с факта, че може да се интерпретира като интерполационна квадратурна формула, но не с алгебрични, а с тригонометрични полиноми. В случая, когато подинтегралната функция може да се сведе до гладка периодична функция, интегрирана за интервал равен на периода, резултат от тип "теорема на Paley-Wiener", дава основания за очакване на експоненциална сходимост (спрямо броя точки в квадратурната формула). Освен това, ако подинтегралната функция има n-та производна с ограничена вариация (V) в интервала, скоростта на сходимост и за гаусовата формула и за тази на Кленшоу-Къртис (за N точки) е една и съща: $O\left(V(2N)^{-n}\right)$. Минало е доста време преди тези неща да се осмислят. Съществуват предубеждения, свързани с ефективността на тази формула, възприемана от гледна точка *само* на алгебрична точност. (вж. например [158] и [159]).

В общи линии и двете използвани схеми дават експоненциална сходимост (когато са приложими) спрямо използвания брой точки в квадратурната формула, но tanh-sinh схемата е с по-голям обхват. Тя, така да се каже, "регуляризира особеностите в краищата на интервала, прехвърляйки ги в безкрайност". От друга страна, там където е приложима, Кленшоу-Къртис схемата е много по-ефективна. От практическа гледна точка това се изразява в това, че ако задачата не се нуждае от такава "регуляризация", броят точки, необходим за пресмятане с предварително зададена точност, е често в пъти по-малък.

За нашата програмна система е реализиран специален графичен програмен инструмент NQTS, както и два конзолни (THSHPar, CCPar). В THSHPar и CCPar се използват паралелни пресмятания. В предпоследния раздел на тази глава са направени някои сравнителни тестове за ефективност. NQTS и THSHPar използват метода на "двойната експонента" (или "двойно експоненциалното преобразование"). CCPar използва квадратурна формула на Clenshaw-Curtis. THSHPar и CCPar са разработени сравнително скоро и при тях, преследвайки максимална ефективност, не е предвиден графичен интерфейс и интерпретатор. Изискваната точност (в брой десетични знаци) и максималното ниво на самата схема се задават от командния ред. При NQTS от друга страна са използвани средства, демонстриращи възможностите за създаване завършен графичен програмен инструмент в използваната от системата среда.

През последните петнайсет-двайсет години пресмятането на определени интеграли с висока точност се превръща в много полезен инструмент за редица области на експерименталната математика. В много случаи е възможно разпознаването в аналитична форма на стойността на даден определен интеграл при условие, че тази стойност може да се изчисли с висока точност. Обикновено за тази цел се използва алгоритъм за определяне (откриване) на целочислена зависимост. Алгоритъмът за определяне на целочислена зависимост (integer relation detection) е числов алгоритъм, който при зададени с определена точност (d десетични знака) n реални числа x_i може да получи целочислените коефициенти a_i, (не всички нули), за които $a_1x_1 + a_2x_2 + ... + a_nx_n = 0$ или в противен случай да определи, че няма такъв n-мерен целочислен вектор, евклидовата норма на който да е по-малка от дадена стойност. Най-често използваният алгоритъм за определяне на целочислена релация е така наречените PSLQ алгоритъм. [22] дава описание на алгоритъма и различни модификации и приложения. Ето прост пример за илюстрация [147]: Даден е интегралът $\int_0^1 \frac{t^2 \ln t}{(t^2-1)(t^4+1)} \mathrm{d}t$. Неговата стойност с точност от 100 десетични знака е

0.18067126259065494279230812898167161533711457101829676626624079429375856622413300177089825415048379 97

Прилагането на схема за определяне на целочислена релация (с PSLQ) дава резултата в "затворена форма" (вж. глава 6 "Метод за определяне на целочислена зависимост и идентификация на константи. Алгоритъм PSLQ").

$$\int_0^1 \frac{t^2 \ln t}{(t^2-1)(t^4+1)}\,\mathrm{d}t = \frac{\pi^2(2-\sqrt{2})}{32}.$$

Между впрочем, специално реализирания програмен инструмент за пресмятане на определени интеграли NQTS (от Numerique quadrature with tanh-sinh transformation) се справя с лекота с подобни интеграли.

Към глава 3. Методи за подобряване на числено решаване на ОДУ.

Както вече беше отбелязано, програмната система е съставена около ядро, надстроявано в различни области. Един от важните стимули за това конкретно разширение е осигуряването на възможност, подпомагаща изследването на интересни нелинейни модели - в случая числено решаване на (системи) ОДУ. Тук на използването на пресмятания с произволна точност може да се погледне от две страни. Първо, формално съществуват схеми, които осигуряват теоретично произволна точност, но реална полза от тях има единствено, ако пресмятанията наистина осигуряват необходимата точност. Второ, и то е по-важното, това дава макар и ограничена възможност за "количествено изследване" в някои случаи, в които то е невъзможно като цяло. Добър пример за това са нелинейните системи, които моделират така наречения "детерминиран хаос". Тук бихме искали да добавим още нещо свързано с известната непълнотата на подхода (ако не се комбинира с други). Исторически първите успехи са в точното решаване, когато е възможно. На друг етап в спиралата в развитието на цялостния поглед това е важно и днес. Отчасти поради важността си в отделни области, отчасти поради това, че често е неизбежна част при по-общо изследване на класове от системи с дадена параметризация. След като става ясно, че това е изключителна възможност, макар и много важна, придобиват тежест и други подходи. Един от тях е възможността за числено решаване. Този подход обаче е с ограничени възможности, поради това, че е приложим за отделни траектории на изследваната система. Освен това постепенно става ясно, че изглеждащите "екзотични" при откритието си модели, пораждащи "детерминиран хаос", са по-скоро правило, отколкото изключение за редица важни области. Оттук следва принципно ново ограничение на подхода с числено решаване. В наши дни е очевидно, че редица въпроси, касаещи поведението на дадена (в нашия случай диференцируема динамична) система изискват други, глобални подходи, отговарящи на въпроси за качественото поведение на различните видове възможни траектории в цялост (пример е КАМ теорията и конкретно теоремата на Колмогоров, Арлнолд и Мозер за това, че в някакъв смисъл повечето тороидални инвариантни подмногообразия във фазовото пространство се запазват при малки смущения в параметрите на напълно интегрируема хамилтоновата система). В процеса на самото числено решаване могат, разбира се, да се отчитат и някои особености на решавания модел от глобално естество. Например това, че естественото пространство, в което еволюира системата, е конкретно многообразие и това да се отчете *пряко* в използвания метод (при избрания от нас основен метод, това става *косвено за определен тип системи*). Въпреки всичко, числеността решаване има своята роля в редица изследвания

и то не само като предварителни "опипващи почвата лабораторни опити" за придобиване на първо впечатление от изучаваната система. Възможността за числено решаване може да се използва като спомагателно средство за друга задача. Един пример за това има в глава 4 "Методи за намиране на корените на скаларни уравнения с висока точност".

Отчитайки всичко това, плюс възприетия "минималистки" принцип, един добър избор на метод е този, реализиращ неявна схема на Рунге-Кута (от произволен ред). Той е симетричен, симплектичен и А-устойчив (за описанието на неявните схеми на Рунге-Кута, както и съответните определения - [49], [40], [75], [160]). Ако става дума само за постигане на висока точност, този метод има много силни конкуренти откъм ефективност. Например прилагащият екстраполация метод на Gragg-Burlish-Stoer, също реализиран в нашата система, или методът с редове на Тейлър. Методът, реализиращ неявна схема на Рунге-Кута, обаче има някои други предимства, поне при решаване на конкретен (но практически много важен) клас системи, а именно - консервативни хамилтонови системи. Методът запазва симплектичната структура на модела. На практика това означава, че продължителна симулация не изкривява основната качествена характеристика на модела заради натрупване на грешка от прекъсване (дискретизация). Все пак:

„Основната идея при проектиране на изчислителни схеми е в това те да могат запазват доколкото е възможно свойствата на оригиналния модел … Различните представяния на един и същ физически закон могат да водят до различни изчислителни техники при решаване на една и съща задача, което може да породи различни числени резултати …" [160].

Нека подчертаем, че разработените програмни инструменти, описвани в тази глава, са посветени най-вече на задачи, които ни се сториха интересни, но далече не покриват всички възможности свързани с числено решаване на с-ми ОДУ. В частност, не се разглеждат задачи, касаещи модели, свързани с управление в реално време.

Първият раздел в тази глава прави изключение в основното изложение - посветен е обсъждане на възможностите и смисъла на решаване на с-ми ОДУ с висока точност за случая, в който те са модел на специален вид системи пораждащи "детерминиран хаос". Пример към него, както и илюстративни решения на някои такива системи са изнесени в отделна глава по-нататък (глава 7). Темата е много обширна и по принцип е свързана с теорията на гладките динамични системи. Малка, но достатъчно представителна част от многобройните източници на информация за първия раздел е [83], [38], [116], [142], [80], [39], [164], [90], [98], [99], [55], [117], [93].

Към глава 4. Намиране на корените на нелинейни скаларни уравнения с много висока точност.

В тази глава са описани възможностите за решаване на нелинейни скаларни (т.е. една променлива) уравнения с висока точност, предоставяни в предлаганата програмна система. Темата е важна и сама за себе си, но е и в помощ на много други пресмятания. Един пример е използването на корените на специални функции в други области на числения анализ. Получените корени се използват например често в асимптотичните представяния на други специални функции, в квадратурни формули от гаусов тип, във физиката (резонанси на механични и електрически системи, задачи на квантовата механика и др.). Темата е достатъчно важна и в контекста на пресмятане с много висока точност. Друг пример е свързан с пресмятанията на някои константи, които след това се използват в алгоритми за определяне на целочислена зависимост (идентификация на константи с алгоритъм PSLQ, примерно). Използваните методи за намиране на корен силно се различават според вида на достъпната първоначална информация за участващата в

уравнението функция и търсения корен. Наличието на добро начално приближение например, позволява използване бързи методи, чиято сходимост обаче, е гарантирана само ако първоначалното приближение е в някаква достатъчно близка околност на корена. В случай че липсва необходимата предварителна информация за корена, трябва да се използват други подходи. Ефективността на намирането на корена (корените) на дадена функция зависи и от предварителната информация за нейните структура, свойства и поведение. Пример за това са ортогоналните полиноми, за които е предварително известно, че корените им са реални, прости (единична кратност) и в определен интервал. Тази информация значително увеличава ефективността на намирането им. Изобщо, при някои специални функции, фактът, че са решение на обикновено диференциално уравнение от определен тип, може да се използва за увеличаване на ефективността при намиране на корени - например при функции на Бесел от първи и втори род. Намирането на корени на уравнения е една от най-старите и добре застъпени теми в числения анализ, в следствие на което броят на публикациите в научни издания, а и монографиите е много голям. Дадените от нас препратки са главно за източници (не всички), които пряко засягат конкретни въпроси при реализацията на методите, в създадените от нас програмни инструменти. Изложението в тази глава е групирано в три раздела. Първите два са за общия случай (не се взема предвид конкретния вид на функцията), съответно при наличие и липса на предварителна информация за корена (наличие на добро начално приближение, информация за кратността). Последният, който е и най-обширен, засяга някои случаи, които използват информацията за структурата, свойствата и поведението на функцията.

За да не го повтаряме многократно по-нататък, изложението в тази глава предполага, че съответните функции са непрекъснати и диференцируеми в околност на изследвания корен, а за случая на глобални методи - в цялата област на определение. Често по-високи производни влизат в оценката за скорост на сходимост. Очевидно в такъв случай самата оценка е валидна за по-гладка функция, макар и в конкретния метод да не се използват по-високи производни. Реализираните в системата локални методи ползват главно първа производна, с изключение на случая на евентуално използване на "доуточняващ" метод (refiner).

Вече беше отбелязано, че почти винаги за решаване на дадена задача има алтернативни методи. Няма да обсъждаме подробно всички възможности, тъй като това би увеличило твърде много текста. Подробността на изложението свързана с реализацията на методите също се различава. За някои са дадени общите идеи. Има случаи обаче, в които без малко по-подробно обяснение, няма да стане ясно как работи методът и защо изобщо работи. На практика изборът на метод е винаги е компромис между ред изисквания между които ефективност, простота (лекота на реализация и поддръжка), универсалност (възможност за използване като самостоятелен блок за вграждане) и т.н. Тези критерии често са противоречиви и изборът на конкретна реализация е по-скоро въпрос на лични предпочитания и виждания за мястото на конкретен метод в програмната система.

В тази част от програмната система има типични места, които очакват "растеж". Например реализация на общите методи за случая n-мерни уравнения.

Към глава 5. Бързо пресмятане на математически константи с много висока точност.

Математическите константи са предмет от особен интерес в пресмятанията с висока точност. Освен че някои от тях се използват често за пресмятания на математически функции (елементарни и специални), те са предварително изискване при реализация на

алгоритъм за разпознаване на целочислена зависимост между реални числа с цел отгатване на аналитичния вид на получен резултат от някаква задача (пример е описаният в глава 6 алгоритъм PSLQ). В специално разработения от нас програмен инструмент MPConsts са отделени методите за пресмятане на някои математически константи. Тук е едно от местата, както се използва реално многоядрената структура на съвременните процесори (другото е паралелната реализация на квадратурни схеми, описана в глава 2).

Максимално бързодействие се постига при наличие на алгоритъм, позволяващ разпаралеляване, с максимално използване на целочислени операции, където е възможно. За редица константи използваме възможността за двоично разделяне (binary splitting). Идеята датира още от [35]. По-съвременни изложения има в [74], а също и в глава 4 на [36].

За някои константи не са известни представяния, позволяващи прилагането на двоично разделяне и се предложени специфични реализации на пресмятане на базата на известни до момента представяния. Всяко от тях е описано на място в тази глава.

Илюстрация на предложената от нас реализация на паралелни пресмятания в конкретната среда на база двоично разделяне е изнесена в отделна глава 8, в която е приведен и списък с представяния на някои константи, които са удобни за прилагане на двоично разделяне.

Към глава 6. Определяне на целочислена зависимост и идентификация на константи. Алгоритъм PSLQ.

Както бе споменато, един от основните мотиви за пресмятане с много висока точност е възможността за идентификация на получения резултат като комбинация от известни математически константи. В крайна сметка, пресмятането с висока точност не е самоцел и разработените в предишните глави средства за такива пресмятания дават възможност за получаване на резултати, чиято точност да е в рамките на изискванията за прилагане на съответния алгоритъм за идентификация. В тази кратка глава е описан един вариант на такъв алгоритъм и резултатите от реализацията му в нашата програмна система, легитимирайки в известна степен извършеното по основната част на предлаганата програмна система.

(История и значение на PSLQ. Експериментална математика)

Определянето (откриването) на целочислена зависимост (integer relation detection) решава следната задача: при дадени n реални числа $x_1, x_2, \ldots, x_n$ да се открие съществуват ли n цели числа (не всички равни на 0) $a_1, a_2, \ldots, a_n$, за които $a_1x_1 + a_2x_2 + \ldots + a_nx_n = 0$. Това равенство, разбира се, при пресмятания с компютър е с точност до "епсилона на използваната точност". В случая $n = 2$ проблемът се решава с пресмятане на непрекъснатата дроб на x_1/x_2. Ако отношението е рационално число, процесът приключва след краен брой стъпки, давайки най-малките a_1 и a_2. В противен случай, след даден брой стъпки има долна граница за размера на тези числа. Т.е. процедурата дава като резултат или съответните цели числа, или долната граница на големината, под която такива не съществуват. Обобщение за случая $n > 2$ е предложено за първи път през 1977 година [61]. Там обаче липсват подробно описание на стъпките, доказателства и граници на точност. Първият завършен алгоритъм е LLL [87], а през 1986 е разработен алгоритъмът HJLS [79]. Имената на алгоритмите са по началните букви на фамилиите на авторите си. През 1988

година Helaman Ferguson публикува алгоритъма PSOS [58] (името е по част от заглавието на публикацията: (P)artial (S)ums (o)f (S)quares). Алгоритъмът PSLQ е разработен през 1992 година [59] и съществено опростен през 1999 година [60]. Името идва от Partial Sums и LQ decomposition. През 2000 година PSLQ е избран за един от "десетте алгоритми на столетието" [51].

Алгоритмите от тип откриване на целочислена зависимост имат многобройни приложения (някои от които ще изброим по-надолу), особено в така наречената "експериментална математика", самото възникване на която предизвиква оживено обсъждане за ролята ѝ в изследванията на "чистата математика" (например [127], където обсъждането касае точно PSLQ). Като начало, според [32] стр. 17:

"Експерименталната математика е този клон на математиката, който по същество се отнася до събирането и разпространяването в математическата общност на прозрения за догадки и неформални убеждения, чрез използване на експериментално изследване (в смисъла влаган от Галилей, Бейкън, Аристотел или Кант) и точен анализ на получените данни от тази дейност".

Към това може да се добави ролята на "използването на съвременните компютърни технологии като активно средство в математическите изследвания" в стил "експериментална математика" [16] стр. 502.

Основните приложения на експерименталната математика с използване на компютри (списъкът по-долу) включва евристики - получаване на по-голямо прозрение и интуитивно разбиране, както и откриване на нови шаблони (схеми) чрез провеждане на експерименти) - (1, 2, 3), уточняване и оценка на догадки с цел дали си струва опит за формално доказателство - (4,5) и помощ в процеса на доказване на догадки - (6, 7, 8).

1. Придобиване на прозрение и интуитивно разбиране.

2. Откриване на нови схеми и съотношения.

3. Използване на графично представяне, подсказващи лежащите в основата математически принципи.

4. Тестване и евентуално опровергаване на варианти на догадки.

5. Изследване на възможен резултат с цел да се види дали си струва формално доказателство.

6. Предложения за подходи за формално доказателство.

7. Заменяне на дълги извеждания на ръка с автоматизирани посредством компютърна програма.

8. Потвърждаване на аналитично изведени резултати.

Без да дава формално доказателство (освен в случая на предоставяне на контрапример), експерименталната математика подпомага пораждането на правдоподобни хипотези, до които трудно се стига по друг начин. Т.е. тя помага в самия творческия процес (основаващ се на интуиция) - този, предхождащ формалното доказателство на дадено твърдение.

Връщайки се на конкретния случай PSLQ, има многобройни примери, в които използването му води до резултати от такъв тип - пораждане на изключително вероятна хипотеза. В компютърната система реалните числа се представят, разбира се, като рационални - първите *d* знака в представянето на съответното число, в зависимост от използваната точност на пресмятанията. Но използвайки достатъчно висока точност (няколко стотин или няколко хиляди знака), намерената евентуално целочислена зависимост е изключително правдоподобна за самите реални числа (неформално "със степен на достоверност" $1 - 10^{-p}$, където p е от порядъка на d, вж. забележките в края на първия раздел, описващ алгоритъма, за това какво всъщност се интерпретира като "ниво на достоверност"). Това, разбира се, не може да замени формалното доказателство, но показва, че такова си струва да се търси. Алгоритъмът изисква междинните пресмятания да се извършват с точност от порядъка на nd, където d е точността, с която се задават числата, а n е броят им. Използването на алгоритми от такъв тип е и една много сериозна причина да се правят пресмятания с висока точност изобщо (практически произволна, в рамките на компютърния ресурс). Един пример за приложение на PSLQ е систематичното изследване с на сумите от вида

$$s_h(m,n) = \sum_{k=1}^{\infty} \left(1 + \frac{1}{2} + \cdots + \frac{1}{k}\right)^m \frac{1}{(k+1)^n}, \quad m \geq 1, n \geq 1,$$

довело с частност през 1996 година до резултат

$$s_h(2,2) = \frac{3}{2}\zeta(4) + \frac{1}{2}\zeta^2(2) = \frac{11\pi^2}{360}$$

доказан малко по-късно аналитично. Интересна е предисторията на изследването, предизвикано от наблюдение на студенти на J. M. Borwein, че

$$\sum_{k=1}^{\infty} \left(1 + \frac{1}{2} + \cdots + \frac{1}{k}\right)^2 \frac{1}{k^2} \approx \frac{17}{4}\zeta(4) = \frac{17\pi^4}{360}$$

с точност от десетина знака. Проверката с PSLQ за 100 знака дава начало на по-общо изследване, включващо споменатия по-горе пример. Много интересни и полезни за практическите пресмятания с висока точност са откритията за разлагане на π в редове от нов тип. Могат естествено да се изследват и други класове задачи - например важни типове интеграли, възникващи в математиката и физиката. Тези и други интересни примери са приведени в [23], [21] и [14]. Друг често срещан проблем е определянето на това, дали дадена константа α е алгебрично число от определена степен n (корен на полином от степен не по-висока от n). Наличието на реализация на PSLQ (или изобщо на някакъв алгоритъм за откриване на целочислена зависимост) ни предоставя веднага очевидна стратегия. Записват се във входния вектор за търсене на целочислена зависимост числата 1, α, α^2, ... , α^n и алгоритъмът намира евентуално коефициентите на полинома, чието решение е α. В случай, че не намери, може да се увеличи n. Конкретен интересен пример са величините $r = B_n$, при които настъпва бифуркация (възниква нов цикъл с по-голяма кратност) на логистичното изображение $x_{k+1} = rx_k(1-x_k)$. Знае се, че те са алгебрични числа, но нищо не е известно предварително за степента и коефициентите на

полиномите, чиито корени са те. За $r = B_3 = 3.54409035955...$ (възникване на цикъл с кратност 8) е намерен полином от 12-та степен, $4913 + 2108t^2 - 604t^3 - 977t^4 + 8t^5 + 44t^6 + 392t^7 - 193t^8 - 40t^9 + 48t^{10} - 12t^{11} + t^{12}$. В случая $r = B_4 = 3.564407266095...$, при който възниква четвъртата бифуркация (възникване на 16-кратен цикъл), неформални разсъждения водят до предположението, че величината $\alpha = -B_4(B_4 - 2)$ би трябвало да е решение на полином от 120-та степен. Такъв е намерен (в [21] има подробности относно използваната точност и достоверността на получения резултат).

Към глава 7. Приложения свързани с пресмятането на системи ОДУ с много висока точност.

В тази глава има два раздела, посветени на примери и илюстрации от решаване на системи с "хаотично поведение". Първият раздел е илюстрация към раздел 3.1 от глава 3. Примерите във втория раздел, представящ илюстрации на решения на прости гладки динамични системи с хаотично поведение, са взети главно от [165]. Освен естетическата си стойност, тези примери послужиха и за своеобразни допълнителни тестове за предлаганите средства за решаване в настоящата програмна система.

Към глава 8. Приложения свързани с бързото пресмятане на някои математически константи

Приведена е илюстрация с програмен код на предложената в настоящата работа реализация на паралелни пресмятания посредством двоично разделяне, а също и списък с представяния на някои константи във вид на редове, които са подходящи за прилагане на двоично разделяне.

Някои възможни насоки за усъвършенстване и разширяване на програмната система.

Тук са групирани на едно място възможностите за бъдещо развитие на програмната система, така както се виждат в настоящия момент.

Цели и рамки.

Самата идеология на настоящата програмна система, вече определя основната цел:

> Даване на възможност на изследователи и разработчици с опит в конкретната среда да създават свой собствен софтуер, използващ пресмятания с произволна точност, за решаване на задачи свързани с експериментална математика.

Целевата аудитория е широка: заинтересуваните от това да решават подобни задачи достатъчно ефективно без да се изисква специален хардуер.

Тук "произволна точност", разбира се, означава произволна в рамките на използвания компютърен ресурс. Това винаги трябва да се има предвид, когато се използва този термин. Когато се споменава "произволна точност", това обикновено значи, че за решавания клас задачи няма априори фиксирана, достатъчна граница за точност.

Стандартен пример е реализация на алгоритъм, осъществяващ идентификация на константа, където работната точност зависи и от това колко математически константи са включени предварително в използваната таблица. Алтернатива, която често е използвана, е терминът "(много) висока точност", макар че това също има минуси, свързани с асоциация за фиксирана точност.

Веднага трябва да се подчертае, че във формулираната по-горе основна цел не влиза създаване на единен интерфейс към всевъзможни програмни инструменти, създадени в рамките на системата, какъвто е предвиден в много среди за математически пресмятания с елементи на компютърна алгебра. Това не е невъзможно по принцип, но има сериозни причини, поради които не се предвижда на настоящия етап. Основната, очевидна, е свързана с евентуалния огромен брой области, за които трябва да има предвидена реализация. Интересните области са много и дори спирайки се само на една, обхващането ѝ в цялост е твърде трудоемка задача. Например, решаването на ОДУ е тема, която далеч не се изчерпва с предоставянето на средства за числено решаване (каквито в нашата система има). Дори да няма други възможности, численото решение не може даде отговори на въпроси, касаещи качественото поведение на решението. Освен това в редица случаи, опитът за символно решаване (теорията на Ли) преди численото е задължителен. Всички тези неща обаче, сами по себе си, изискват отделни програмни инструменти, чиято разработка надхвърля рамките на нашата система. Тя е съсредоточена до момента главно в реализация на методи, посволяващи решаване на задачи на числения анализ в контекст на използване на получените с много висока точност резултати за целите на експерименталната математика. Направени са някои улеснения, свързани с интероперативността на някои програмни инструменти, но това не е общ подход, а резултат от опита ни, диктуващ известни улеснения при използване на тези често използвани програмни инструменти. В крайна сметка, а това е и друга причина, идеологията на системата предвижда да бъде нещо като нарастващ организъм, вероятно по различен начин при използването и от различни нейни потребители. Направено е обаче усилие да се покажат силните страни на използваната среда от чисто програмистка гледна точка. Налице са нагледни примери за лесната интеграция на различни методи с възможностите на системата за построяване на интерактивен графичен интерфейс, както и използване на предоставяните от средата отлични възможности за паралелни пресмятания (в частност Task Parallel Library на .Net Framework).

Въпреки че разработените програмни инструменти в предлаганата система са предимно в рамките на числения анализ, условно обслужващ (с много високата точност) експериментална математика и тук има известни ограничения. С малки изключения, не се предвиждат програмни инструменти за една и съща задача, използващи множество различни методи. Използвано е това, което сме решили, че си струва. Изборът почти винаги е свързан с предварителна обработка на много информация, а често и с предварителни тестове, за да се вземе решението, кое е перспективно. Освен това възможността за добавяне на всевъзможни нови програмни инструменти или разширяване на съществуващите с други методи си остава. Като цяло, отчитайки всички споменати вече субективни елементи, изведената по-горе в каре основна цел може да се конкретизира и детайлизира по следния начин:

Да се създадат средства за разработка на програмни инструменти, работещи с произволна точност, в конкретната програмна среда (.Net Framework), които да могат да се интегрират непосредствено в процеса на разработката.

Да се реализират конкретни програмни инструменти в областта на числения анализ, подпомагащи изследвания в областта на експерименталната математика, използващи създадените за целта средства.

Да се изследва и по възможност реализира възможността за използване на паралелни пресмятания, предоставяна от многоядрената архитектура на съвременните процесори, вграждани в масовите настолни и преносими компютри.

Изпълнението на задачите в тази конкретизация на общата цел, формулирана по-горе, би доказала, че тя е *възможна*, *полезна* и *ефективна*. Целта е достатъчно амбициозна, като се има предвид, че на повечето етапи се изисква

разработване на методи, които да отговарят на спецификата на създаваната система, а именно да са достатъчно ефективни при пресмятане с произволна точност.

Тъй като акцентът навсякъде е върху пресмятания с произволна точност, под ефективност обикновено се разбира такава, свързана с времето за решаване, съотнесено към изискваната точност и евентуалната размерност на задачата.

Глава 1. Методи и средства за подобряване на пресмятанията на елементарни и специални функции.

1.1. Някои методи за пресмятане.

За елементарните функции съществуват някои общи техники, които са полезни на много места. Една от най-често използваните е редуциране на аргумента, което е следствие от конкретни формули за събиране или симетрии, допълнително съкращаващи броя на изискваните за пресмятане членове на някакъв бързо сходящ ред (ако се използва такъв). Конкретни примери:

За експоненциалната функция например може да се използва $\exp(x) = \exp\left(\frac{x}{2}\right)^2$, а за арктангенса $\arctan(x) = 2\arctan\left(\frac{\sqrt{1+x^2}-1}{x}\right), \quad x > 0$.

При отрицателни стойности на x би трябвало да се пресмята $1/\exp(-x)$ вместо $\exp(x)$ за да се избегне възможна загуба на точност от съкращаване на членове. Известно внимание изисква и възможната загуба на точност при редукция. Например, ако използваме $\exp(x) = \exp\left(\frac{x}{2^k}\right)^{2^k}$, $x/2^k$ е $O(2^{-k})$ и при последователното повдигане на степен (2^k пъти) на реда $(1+x/2^k+...)$ губим поне k бита точност. Това може да се предотврати, ако използваме функцията $\mathrm{expm1}(x) = \exp(x) - 1$, за която формулата за удвояване на аргумента е $\mathrm{expm1}(2x) = \mathrm{expm1}(x)(2 - \mathrm{expm1}(x))$. Впрочем, вместо да се пресмята exp непосредствено с $\sum_{k=0}^{\infty} \frac{x^k}{k!}$, може да се пресметне $\sinh(x) = \sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)!}$ и да се използва $\exp(x) = \sinh(x) + \sqrt{1+\sinh^2(x)}$, съкращавайки броя на изискваните за дадена точност членове приблизително наполовина.

При тригонометричните функции замяната $\xi = \frac{x}{\pi}, \quad m = \lfloor \xi + 1/2 \rfloor, \quad \theta(\xi - m)$ дава $\sin(x) = (-1)^m \sin(\theta)$ и $\cos(x) = (-1)^m \cos(\theta)$, с $|\theta| \le \frac{\pi}{2}$. По-нататъшна редукция може да се осъществи с някоя от формулите за кратен аргумент, например $\cos(2x) = 2\cos^2(x) - 1$. Един възможен вариант на алгоритъм пресмятащ косинус по тази схема с точност p бита изглежда така

Вход: x, p /* аргумент и изисквана точност */

Изход: $\cos(x)$ с p-битова точност.

$x = |x|$

Пресмятат се ξ, m както по-горе

$k = \lfloor \sqrt{n/2} \rfloor$

$\eta = p + k$

/* Задава се работна точност prec > p + k, 2p е предостатъчно */

prec = 2p

$r = \xi^2 / 2^{2k}$

$s = 1, t = 1, l = 1$

do

$t = t * r$

$t = \frac{t}{(2l-1)2l}$

$s = s + (-1)^l t$

while $|t| \geq \eta$

for j = 1 to k do

$s = 2s^2 - 1$

enddo

if m%2 != 0

s = -s

endif

return s

Фигура 1.1. Схема за пресмятане на cos с редукция на аргумента.

За естествения логаритъм също може да се използва редукция. Праволинейното използване на ред на Тейлър за ln(1+x) не е ефективно, но представянето

$\ln\left(\frac{1+y}{1-y}\right) = 2\sum_{k=0}^{\infty} \frac{y^{2k+1}}{2k+1}$ с y определено от (1+y)/(1-y) = 1+x дава y = x(2+x). Това спестява около половината членове за пресмятане, т.к. свежда аргумента до y < x/2 (при x > 0).

В общи линии пресмятането със степенни редове от вида $\sum_{k=0}^{\infty} a_k x^{\alpha k+\beta}$ е изгодно да се прави в случая, когато отношението $\frac{a_{k+1}}{a_k} = R(k)$ е някаква рационална функция на k и следователно изчисленията могат да се организират по такъв начин, че за всеки следващ член (освен умножението по x^{α}) се изискват целочислени операции и умножение и/или деление на цяло число на предишния член, което е значително по-ефективно реализирано

дори в аритметика с плаваща запетая. Тъй като елементарните функции (и доста голям брой специални, които са решение на някакъв частен случай на хипергеометричното уравнение) са частни случай на хипергеометрична функция, редовете, с които се представят изпълняват автоматично това условие.

Нека $M(d)$ означава времето, необходимо за умножение на числа с d-битова точност (като се предполага, че времето за останалите операции е много по-малко и следователно пренебрежимо). При описаните по-горе условия, ако е необходимо да се пресметне степенен ред $\sum_{k=0}^{\infty} a_k x^{\alpha k}$, чийто радиус на сходимост е поне единица, с точност $O(2^{-d})$ и $|x| < \frac{1}{2}$, достатъчно е да се пресметнат $d + O(1)$ членове, т.е. необходимото време е $O(dM(d))$. Прилагайки редукция $k + O(1)$ пъти, може да се осигури $|x| < 2^{-k}$. Тогава необходимият брой членове за пресмятане е $O\left(\frac{d}{k}\right)$. Следователно времето, което се изисква за постигане на точност 2^{-d} е $O\left(\left(\frac{d}{k}+k\right)M(d)\right)$. Тук се предполага, че времето за една стъпка на редукцията е от порядъка $M(d)$, което е вярно за елементарните функции. Ако изберем $k \sim \sqrt{d}$, получената оценка е $O\left(d^{1/2}M(d)\right)$.

В някои случаи обаче, ако изискваната точност d е голяма, могат да се използват и по-ефективни методи. Най-бързите от тях са свързани с пресмятане на аритметично-геометрично средно (AGM итерация) и оценката им е $O(M(d)\log(d))$. Скритата константа тук обаче е такава, че за сравнително малки d има по-добри методи, включително използващи редове. При дадени начални числа a_0, b_0 AGM итерацията се определя с $\left(a_{n+1}, g_{n+1}\right) = \left(\frac{a_n + g_n}{2}, \sqrt{a_n g_n}\right)$. При предположение, че a_0 и g_0 са реални и положителни (може да се обобщи за комплексни числа, вж. например [30]), AGM итерацията е квадратично сходяща към граница, означавана с $\mathrm{AGM}(a_0, g_0)$. Тъй като квадратичната сходимост удвоява на итерация броя на верните десетични знаци, изискваният брой итерации е $O(\log(d))$. При подходяща реализация за пресмятане на квадратния корен (както е гарантирано при нас например от MPIR), времето за всяка итерация е $M(d)$. При подходящ избор на началните стойности a_0, g_0 може да се пресмятат логаритми, елиптични интеграли и някои константи.

Тясната връзка на AGM итерацията с някои видове елиптични интеграли е забелязана от Гаус още в 1799 г. Изчисленията му показват, че $\frac{1}{AGM(1,1/\sqrt{2})}$ и $\frac{2}{\pi}\int_0^1 \frac{dt}{\sqrt{1-t^4}}$ се съгласуват с поне единайсет знака точност. По-късно доказва [67] общия резултат

$$\frac{1}{AGM(1,x)} = \frac{2}{\pi}\int_0^{\pi/2} \frac{d\theta}{\sqrt{1-(1-x^2)\sin^2\theta}} \qquad (1.1)$$

Пълните елиптични интеграли от първи и втори род се определят съответно с

$$K\left(k\right)=\int_0^{\frac{\pi}{2}}\frac{d\theta}{\sqrt{1-k^2\sin^2\theta}}=\int_0^1\frac{dt}{\sqrt{(1-t^2)(1-k^2t^2)}} \tag{1.2}$$

и

$$E\left(k\right)=\int_0^{\frac{\pi}{2}}\sqrt{1-k^2\sin^2\theta}\,d\theta=\int_0^1\sqrt{\frac{1-k^2t^2}{1-t^2}}\,dt\,. \tag{1.3}$$

$k\in[0,1]$ се нарича модул, а k' е допълнителният (допълващият) модул (complementary modulus), се определя от $k^2+k'^2=1$. Имайки това предвид, (1) може да се запише и като

$$K(k')=\frac{\pi}{2}\frac{1}{AGM(1,k)}. \tag{1.4}$$

Ако се определи $c_n=\sqrt{{a_n}^2-{g_n}^2}$, то

$$E\left(k\right)=K\left(k\right)\left[{a_1}^2-\sum_{n=2}^{\infty}2^{n-1}{c_n}^2\right]. \tag{1.5}$$

От друга страна, $K(k)$ може да се представи със сходящ (за $\left|k\right|<1$) ред, тъй като

$$K(k)=\frac{\pi}{2}\,{}_2F_1\left(\frac{1}{2},\frac{1}{2};1;k^2\right)=\frac{\pi}{2}\sum_{n=0}^{\infty}\frac{(1/2)_n(1/2)_n}{n!n!}k^{2n}=\frac{\pi}{2}\sum_{n=0}^{\infty}\left[\frac{(2n-1)!!}{2^n n!}\right]^2k^{2n},$$ за малки k е изпълнено

$$K(k)=\frac{\pi}{2}\left(1+\frac{k^2}{4}\right)+O(k^4), \tag{1.6}$$

а тъй като ([30])

$$K(k')=\frac{2}{\pi}\ln\left(\frac{4}{k}\right)K(k)+O(k^4), \tag{1.7}$$

получаваме в крайна сметка

$$\frac{\pi/2}{AGM(1,k)}=\ln\left(\frac{4}{k}\right)\left(1+O(k^2)\right). \tag{1.8}$$

За достатъчно големи стойности $x=4/k\geq 2^d$

$$\ln(x)=\frac{\pi/2}{AGM(1,4/x)}. \tag{1.9}$$

За пресмятането на π има разбира се много начини, но също може да се използва AGM итерация (така наречената формула на Brent-Salamin [119]). Ако при начални стойности $a_0 = 1, b_0 = 1/\sqrt{2}$ в процеса на AGM итерациите пресмятаме също и $c_{n+1} = (1/2)(a_n - b_n)$,

$$\pi = 4\frac{\left[AGM\left(1,1/\sqrt{2}\right)\right]^2}{1-\sum_{k=1}^{\infty} 2^{k+1} c_k}. \tag{1.10}$$

Ако *x* не е достатъчно голямо, може да де използва равенството (като се предполага независимо пресмятане на ln2) $\ln\left(2^k x\right) = k\ln(2) + \ln(x)$ или (при $x > 1$)

$$\ln\left(x^{2^k}\right) = 2^k \ln(x), \tag{1.11}$$

което при $x = 2$ е начин за пресмятане на ln2 . Тук обаче възникват известни проблеми с контрола на грешката (за $k > 0.5$). Има точна формула, при която такъв проблем не възниква (формулата на Sasaki-Kanada [121], малко по-надолу). За самото ѝ разбиране обаче, първо трябва да се въведат така наречените тета-функции, които са тясно свързани с AGM итерацията (оттам и с пресмятането на пълните елиптични интеграли от първи и втори род). В общия си вид те са функции на две комплексни променливи z, τ, като зависимостта от втората е чрез $q = e^{i\pi\tau}$, $\mathrm{Im}(\tau) > 0$.

$$\theta_1(z\,|\,\tau) = \theta_1(z,q) = 2\sum_{n=0}^{\infty} (-1)^n q^{\left(n+\frac{1}{2}\right)^2} \sin\left((2n+2)z\right), \tag{1.12}$$

$$\theta_2(z\,|\,\tau) = \theta_2(z,q) = 2\sum_{n=0}^{\infty} q^{\left(n+\frac{1}{2}\right)^2} \cos\left((2n+2)z\right), \tag{1.13}$$

$$\theta_3(z\,|\,\tau) = \theta_3(z,q) = 1 + 2\sum_{n=1}^{\infty} q^{n^2} \cos\left(2nz\right), \tag{1.14}$$

$$\theta_4(z\,|\,\tau) = \theta_4(z,q) = 1 + 2\sum_{n=1}^{\infty} (-1)^n q^{n^2} \cos\left(2nz\right) \tag{1.15}$$

В този си вид те се използват например за пресмятането на елиптичните функции на Якоби. В този момент ще използваме един друг вариант (базови тета функции) с една променлива q, за която областта на определение е $|q| < 1$ ($z = 0$).

$$\theta_2(q) = \sum_{n=-\infty}^{\infty} q^{(n+1/2)^2} = 2q^{1/4}\sum_{n=0}^{\infty} q^{n(n+1)} \tag{1.16}$$

(1.17) $$\theta_3(q)=\sum_{n=-\infty}^{\infty}q^{n^2}=1+2\sum_{n=1}^{\infty}q^{n^2}$$

(1.18) $$\theta_4(q)=\theta_3(-q)=1+2\sum_{n=1}^{\infty}(-1)^n q^{n^2}$$

Като се възползваме от равенствата ([30]) $\left(\theta_3^2(q)+\theta_4^2(q)\right)\big/2=\theta_3^2\left(q^2\right)$ и $\theta_3(q)\theta_4(q)=\theta_4^2\left(q^2\right)$, което може да се запише и като $\sqrt{\theta_3^2(q)\theta_4^2(q)}=\theta_4^2\left(q^2\right)$, установяваме връзката с AGM:

(1.19)
$$AGM\left(\theta_3^2(q),\theta_4^2(q)\right)=\cdots=AGM\left(\theta_3^2\left(q^{2^k}\right),\theta_4^2\left(q^{2^k}\right)\right)=\cdots=1\,,|q|<1\,.$$

Мащабирането на $AGM\left(\theta_3^2(q),\theta_4^2(q)\right)=1$ ни дава

(1.20) $$AGM(1,k')=\frac{1}{\theta_3^2(q)},$$

при условие, че $k'=\theta_4^2(q)\big/\theta_3^2(q)$ или, което е еквивалентно (поради това, че $\theta_2^4(q)+\theta_4^4(q)=\theta_3^4(q)$)

(1.21) $$k=\theta_2^2(q)\big/\theta_3^2(q)\,.$$

Ние обаче знаем, че $K(k)=\dfrac{\pi/2}{AGM(1,k')}$, така че $K(k)=\dfrac{\pi}{2}\theta_3^2(q)$.

Връзката между k и q, може да се обърне и се дава с

(1.22) $$q=\exp\left(-\pi\frac{K(k')}{K(k)}\right),$$

което е еквивалентно на

(1.23) $$\ln\left(\frac{1}{q}\right)=\pi\frac{K(k')}{K(k)}$$

Обединявайки всичко до тук, получаваме формулата на Sasaki-Kanada

(1.24) $$\ln\left(\frac{1}{q}\right)=\frac{\pi}{AGM(\theta_2^2(q),\theta_3^2(q))}$$

Тази формула е в основата на много ефективен алгоритъм. Като заменим във формулата q с q^4, за да избегнем $q^{1/4}$ в определението на θ_2 можем да изберем $x = 1/q$

достатъчно голямо (възползвайки се от (10)), за да бъдат в θ_2 и θ_3 пренебрежими членовете $O(q^{36})$, например Това значи, че вземайки в

$$\ln(x) = \frac{\pi}{AGM(\theta_2^2(q^4), \theta_3^2(q^4))} \tag{1.25}$$

$x > 2^{d/36}$ в съответните редове пресмятаме само

$$\theta_2(q^4) = 2(q + q^9 + q^{25}) + O(q^{49})$$

и

$$\theta_3(q^4) = 1 + 2(q^4 + q^{16}) + O(q^{36}).$$

Освен това, коренът в AGM итерацията може да бъде избегнат, тъй като

$$AGM(\theta_2^2, \theta_3^2) = \left(\frac{\theta_2^2 + \theta_3^2}{2}, \frac{2\theta_2\theta_3}{2} \right) = \frac{AGM(\theta_2^2 + \theta_3^2, 2\theta_2\theta_3)}{2}.$$

Гама функция.

За аргумент x+1 в интервала [1,2) Гама функцията се пресмята по следния начин [128]:

$$x! = \Gamma(x+1) = (x+a)^{x+\frac{1}{2}} e^{-(x+a)} \sqrt{2\pi} \left[c_0 + \sum_{k=1}^{a-1} \frac{c_k}{x+k} + \varepsilon(x) \right]. \tag{1.26}$$

Тук a е положителен параметър, контролиращ точността (вж. по-долу), а коефициентите c_k се определят по следния начин:

$$\begin{cases} c_0 = 1 \\ c_k = \dfrac{1}{\sqrt{2\pi}} \dfrac{(-1)^{k+1}}{(k-1)!} (-k+a)^{k-\frac{1}{2}} e^{-k+a}, \quad k = 1, 2, \ldots, a-1. \end{cases} \tag{1.27}$$

Относителната грешка при x > 0 и a > 2 е $\varepsilon \le a^{-\frac{1}{2}} (2\pi)^{-\left(a+\frac{1}{2}\right)}$, така че след изпускане на първия множител (което завишава оценката), получаваме

$$a = -\frac{\log(\varepsilon)}{\log(2\pi)} \approx -1{,}84 \log(\varepsilon). \tag{1.28}$$

Ако искаме точност от n значещи цифри, то $\varepsilon = 10^{-n}$ и следователно

$$a = \frac{1}{\log(2\pi)} n \approx 1{,}26n \tag{1.29}$$

При отрицателен аргумент се използва Γ(x) = π /[sin(π *x)* Γ(1 - x)]

За положителен аргумент:

Ако x < 1 се използва Γ(x) = Γ(x+1)/x. Ако x ≥ 2, то x = y + n, където y е в [1,2) и Γ(x) = (y + n - 1) … (y + 1)y Γ(y)

Непълни гама функции, вероятностни интеграли.

За

$$\begin{cases} \gamma(s,x)=\int_0^x t^{s-1}e^{-t}dt, \\ \Gamma(s,x)=\int_x^\infty t^{s-1}e^{-t}dt, \end{cases} \tag{1.30}$$

са използвани са стандартни редове и асимптотични разложения.

Стандартното представяне е $\gamma(s,x)=x^s s^{-1}{}_1F_1(s;s+1;-x)$,

а асимптотичното представяне

$$\Gamma(s,x)=x^{s-1}e^{-x}\left(1+\frac{s-1}{x}+\frac{(s-1)(s-2)}{x^2}+\dots\right). \tag{1.31}$$

$\gamma(0,x)$ не е определено, но при $s\to 0$ $\Gamma(s,x)$ има граница

(1.32)

$$\Gamma(0,x)=\lim_{s\to 0}\left[\Gamma(s)-\frac{1}{s}-\left(\gamma(s,x)-\frac{1}{s}\right)\right]=-\gamma-\ln(x)-\sum_{n=1}^{\infty}(-1)^n\frac{x^n}{n(n!)}.$$

Към момента не е включено асимптотичното представяне по първия аргумент.

За

$$\begin{cases} erf(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt, \\ erfc(x)=\frac{2}{\sqrt{\pi}}\int_x^\infty e^{-t^2}dt, \\ erf(x)+erfc(x)=1. \end{cases} \tag{1.33}$$

се използва се представянето

$$erf(x)=\frac{2}{\sqrt{\pi}}\sum_{n=0}^{\infty}\frac{(-1)^n x^{2n+1}}{n!(2n+1)}, \tag{1.34}$$

а за голяма стойност на аргумента

$$erfc(x) \sim \frac{e^{-x^2}}{\sqrt{\pi}\,x} \sum_{n=0}^{\infty} (-1)^n \frac{1.3.\cdots(2n-1)}{(2x^2)^n}. \tag{1.35}$$

Интеграли на Френел.

Използвани са стандартни редове и асимптотични разложения.

За изчисляване на интегралите на Френел

$$C(x) = \int_0^x \cos\left(\frac{1}{2}\pi t^2\right) dt,\ S(x) = \int_0^x \sin\left(\frac{1}{2}\pi t^2\right) dt \tag{1.36}$$

при неголям аргумент се използват представяния с редовете

$$\begin{cases} C(x) = \cos\left(\frac{1}{2}\pi x^2\right) \sum_{n=0}^{\infty} \frac{(-1)^n \pi^{2n}}{1.3.\ldots.(4n+1)} x^{4n+1} + \\ \quad + \sin\left(\frac{1}{2}\pi x^2\right) \sum_{n=0}^{\infty} \frac{(-1)^n \pi^{2n+1}}{1.3.\ldots.(4n+3)} x^{4n+3}, \end{cases} \tag{1.37}$$

$$\begin{cases} S(x) = -\cos\left(\frac{1}{2}\pi x^2\right) \sum_{n=0}^{\infty} \frac{(-1)^n \pi^{2n+1}}{1.3.\ldots.(4n+3)} x^{4n+3} + \\ \quad + \sin\left(\frac{1}{2}\pi x^2\right) \sum_{n=0}^{\infty} \frac{(-1)^n \pi^{2n}}{1.3.\ldots.(4n+1)} x^{4n+1}. \end{cases} \tag{1.38}$$

За асимптотичното представяме се използват редовете

$$C(x) = \frac{1}{2} + f(x)\sin\left(\frac{1}{2}\pi x^2\right) - g(x)\cos\left(\frac{1}{2}\pi x^2\right), \tag{1.39}$$

$$S(x) = \frac{1}{2} - f(x)\cos\left(\frac{1}{2}\pi x^2\right) - g(x)\sin\left(\frac{1}{2}\pi x^2\right), \tag{1.40}$$

където

$$f(x) \sim \frac{1}{\pi x} \sum_{n=0}^{\infty} (-1)^n \frac{1.3.\ldots.(4n-1)}{(\pi x^2)^{2n}} \tag{1.41}$$

(1.42) $$g(x) \sim \frac{1}{\pi^2 x^3} \sum_{n=0}^{\infty} (-1)^n \frac{1.3.\ldots.(4n+1)}{(\pi x^2)^{2n}}.$$

Функции на Бесел от първи и втори род ,модифицирани беселови функции, функции на Ейри [140].

Използвани са стандартни редове и асимптотични разложения. Основното представяне е

(1.43) $$J_\nu(x) = \frac{\left(\frac{1}{2}x\right)^\nu}{\Gamma(\nu+1)} {}_0F_1\left(;\nu+1;-\frac{1}{4}x^2\right).$$

Асимптотичното представяне е

(1.44)

$$J_\nu(x) \sim \left(\frac{2}{\pi x}\right)^{\frac{1}{2}} \left[\cos(\omega) \sum_{k=0}^{\infty} (-1)^k \frac{a_{2k}(\nu)}{x^{2k}} - \sin(\omega) \sum (-1)^k \frac{a_{2k+1}(\nu)}{x^{2k+1}}\right],$$

където

(1.45) $$\begin{cases} a_0(\nu) = 1, \\ a_k(\nu) = \dfrac{\left(4\nu^2 - 1^2\right)\left(4\nu^2 - 3^2\right)\ldots\left(4\nu^2 - (2k-1)^2\right)}{k! 8^k}, \quad k \geq 1, \\ \omega = x - \left(\dfrac{2\nu+1}{4}\right)\pi. \end{cases}$$

Функциите на Бесел от втори род за нецелочислен индекс се изразяват чрез тези от първи род с

(1.46) $$Y_\nu(x) = \frac{J_\nu(x)\cos(\nu\pi) - J_{-\nu}(x)}{\sin(\nu\pi)}.$$

За целочислен индекс представянето е

$$
\begin{aligned}
Y_n(x) \;&= -\frac{\left(\frac{1}{2}x\right)^{-n}}{\pi}\sum_{k=0}^{n-1}\frac{(n-k-1)!}{k!}\left(\frac{1}{2}x^2\right)^k \\
&\quad + \frac{2}{\pi}\ln\left(\frac{x}{2}\right)J_n(x) \\
&\quad - \frac{\left(\frac{1}{2}x\right)^{n}}{\pi}\sum_{k=0}^{\infty}\Big[\psi(k+1)+\psi(n+k+1)\Big]\frac{\left(-\frac{1}{2}x^2\right)^k}{k!(n+k)!},
\end{aligned}
\tag{1.47}
$$

където ψ е логаритмичната производна на гама функцията. За положителни целочислени стойности

$$
\begin{cases}
\psi(1) = -\gamma, \\
\psi(n) = -\gamma + \sum_{k=1}^{n-1}\frac{1}{k}, \quad k \geq 2.
\end{cases}
\tag{1.48}
$$

Тук γ е константата на Ойлер $\gamma = \lim_{n\to\infty}\left(\sum_{k=1}^{n}\frac{1}{k} - \ln(n)\right)$.

Асимптотичното представяне е

(1.49)

$$
Y_\nu(x) \sim \left(\frac{2}{\pi x}\right)^{\frac{1}{2}}\left[\sin(\omega)\sum_{k=0}^{\infty}(-1)^k\frac{a_{2k}(\nu)}{x^{2k}} + \cos(\omega)\sum_{k=0}^{\infty}(-1)^k\frac{a_{2k+1}(\nu)}{x^{2k+1}}\right]
$$

и означенията са същите като по-горе.

За модифицираните функции на Бесел *I* съответно за малък и голям аргумент:

$$
I_\nu(x) = \frac{\left(\frac{1}{2}x\right)^{\nu}}{\Gamma(\nu+1)}{}_0F_1\left(;\nu+1;\frac{1}{4}x^2\right)
\tag{1.50}
$$

$$
I_\nu(x) \sim \frac{e^x}{\sqrt{2\pi x}}\sum_{n=0}^{\infty}(-1)^n\frac{a_{n\backslash}(\nu)}{x^n}
\tag{1.51}
$$

с предишните означения за асимптотиката.

За модифицираните функции на Бесел *K*, за нецелочислен индекс:

(1.52) $$K_\nu(x)=\frac{1}{2}\pi\frac{I_{-\nu}(x)-I_\nu(x)}{\sin(\nu\pi)},$$

за целочислен индекс:

(1.53)
$$\begin{aligned}K_n(x) &= \frac{1}{2}\left(\frac{x}{2}\right)^{-n}\sum_{k=0}^{n-1}\frac{(n-k-1)!}{k!}\left(\frac{1}{2}x^2\right)^k \\ &+\left(-1\right)^{n+1}\ln\left(\frac{x}{2}\right)I_n(x) \\ &+\left(-1\right)^n\left(\frac{x}{2}\right)^n\sum_{k=0}^{\infty}\left[\psi(k+1)+\psi(n+k+1)\right]\frac{\left(\frac{1}{2}x^2\right)^k}{k!(n+k)!},\end{aligned}$$

а асимптотичното представяне е $K_\nu(x)\sim\sqrt{\frac{\pi}{2x}}e^{-x}\sum_{n=0}^{\infty}\frac{a_n(\nu)}{x^n}$, с предишните означения.

Към момента не е включено асимптотичното представяне за големи стойности на индекса.

За функциите на Ейри използваме изразяване чрез модифицираната функция на Бесел.

За x = 0,

(1.54)
$$\begin{cases}Ai(0)=\dfrac{1}{3^{2/3}\Gamma\left(\frac{2}{3}\right)}, \\ Bi(0)=\dfrac{1}{3^{1/6}\Gamma\left(\frac{2}{3}\right)}.\end{cases}$$

за x > 0,

(1.55)
$$\begin{cases}Ai(x)=\dfrac{1}{\pi}\sqrt{\frac{x}{3}}K_{1/3}\left(\frac{2}{3}x^{3/2}\right), \\ Bi(x)=\sqrt{\frac{x}{3}}\left[I_{1/3}\left(\frac{2}{3}x^{3/2}\right)+I_{-1/3}\left(\frac{2}{3}x^{3/2}\right)\right].\end{cases}$$

за x < 0,

$$\begin{cases} Ai(-x)=\dfrac{\sqrt{x}}{3}\left[J_{1/3}\left(\dfrac{2}{3}x^{3/2}\right)+J_{-1/3}\left(\dfrac{2}{3}x^{3/2}\right)\right], \\ Bi(-x)=\sqrt{\dfrac{x}{3}}\left[J_{-1/3}\left(\dfrac{2}{3}x^{3/2}\right)-J_{1/3}\left(\dfrac{2}{3}x^{3/2}\right)\right]. \end{cases} \tag{1.56}$$

Зета функция на Риман [137], [56], [82].

Използван е Алгоритъм 3 от публикацията на P. Borwein [27].

ζ - функцията на Риман е аналитично продължение на

$$\begin{cases} \zeta(s)=\sum_{n=1}^{\infty}\dfrac{1}{n^s}, \mathrm{Re}(s)>1 \\ \quad =\dfrac{1}{\left(1-2^{1-s}\right)}\sum_{n=1}^{\infty}\dfrac{(-1)^{n+1}}{n^s}, \mathrm{Re}(s)>0. \end{cases} \tag{1.57}$$

Алгоритъмът се състои в следното:

Нека определим

$$e_j=(-1)^j\left[\sum_{k=0}^{j-n}\frac{n!}{k!(n-k)!}-2^n\right], \tag{1.58}$$

където празна сума се счита за нула. Тогава

$$\zeta(s)=\frac{-1}{2^n\left(1-2^{1-s}\right)}\sum_{j=0}^{2n-1}\frac{e_j}{(j+1)^s}+\gamma_n(s), \tag{1.59}$$

където за Re(s) > 0

$$\left|\gamma_n(s)\right|\le\frac{1}{8^n}\frac{\left(1+\left|\dfrac{\mathrm{Im}(s)}{\mathrm{Re}(s)}\right|\right)e^{\frac{\left|\mathrm{Im}(s)\right|\pi}{2}}}{\left|1-2^{1-s}\right|}, \tag{1.60}$$

а за Re(s) в [–(n-1) , 0)

$$\left|\gamma_n(s)\right|\le\frac{4^{\left|\mathrm{Re}(s)\right|}}{8^n\left|1-2^{1-s}\right|\left|\Gamma(s)\right|}. \tag{1.61}$$

В случая на реален аргумент оценката на остатъчния член става по-проста за употреба.

Числа на Бернули.

За ефективното им пресмятане заслужават внимание поне три метода, един от които е за случая, когато се изискват всички до определен номер (описан е на мястото, където се използва специално, в глава 5). От другите два, единият е реализиран в основната библиотека и описан по-долу. Третият, най-ефективен в случай на пресмятане на единствено число на Бернули с голям индекс (субквадратичен, [77]) не е реализиран. Ефективната му реализация е доста сложна и в момента няма критична необходимост от него в други части на програмната система.

Използва се метод описан в [94].

Алгоритъмът се свежда до следното:

При предположение, че $m \geq 2$ е четно, пресмятаме последователно

1. $K = \dfrac{2m!}{\left(2\pi\right)^m}$
2. $d = \prod_{p-1|m} p$
3. $N = \left\lceil \left(Kd\right)^{1/(m-1)} \right\rceil$
4. $z = \prod_{p \leq N} \left(1-p\right)^{-1}$
5. $a = \left(-1\right)^{m/2+1} \left\lceil dKz \right\rceil$
6. $B_m = \dfrac{a}{d}$

Фигура 1.2. Алгоритъм за пресмятане на число на Бернули.

В 2 произведението е за всички прости числа *p*, за които *p* - 1 дели *m*. Съответно в 4 произведението е за простите числа по-малки или равни на *N*. В първата стъпка *K* трябва да е изчислено с достатъчна точност, така че пресмятането в 5 да даде желания резултат. За стойност на *N* може да се вземе всяко цяло число, което е по-голямо или равно на това определено в 3.

Да обясним накратко защо това работи. Този алгоритъм използва следните резултати.

Първо (произведението е по простите числа),

(1.62) $$\zeta\left(s\right) = \prod_p \left(1-p^{-s}\right)^{-1}.$$

Второ, за всяко цяло число $m \geq 1$

(1.63) $$2\zeta\left(2m\right) = \frac{\left(-1\right)^{m+1}\left(2\pi\right)^{2m}}{\left(2m\right)!} B_{2m},$$

откъдето получаваме за всяко четно цяло число $m \geq 4$

(1.64) $$\left|B_m\right| = \frac{2m!}{\left(2\pi\right)^m}\zeta\left(m\right).$$

Освен тези два резултата, доказани от Ойлер, има и теорема на von Staudt и Clausen (преоткрита от Рамануджан, вж. съответните теми например в [144] и [143], първоизточникът [131] от 1840 г. е труднодостъпен), която описва делителя на B_m (представено като деление на взаимно прости цели числа) чрез делителите на *m*. Това е произведението в стъпка 2 на алгоритъма. В стъпка 1 на алгоритъма K е определено така, че $\left|B_m\right| = K\zeta\left(m\right)$. Използвайки (59) можем да апроксимираме $\zeta\left(m\right)$ отдолу с произволна точност. Ако е пресметнато число *z*, за което $0 \leq \zeta\left(m\right) - z < \left(Kd\right)^{-1}$, то $0 \leq \left|B_m\right| - zK < d^{-1}$ и следователно $0 \leq \left|a\right| - zKd < 1$. С $\left|a\right|$ е означен числителят на $\left|B_m\right|$. От тук следва, че $\left|a\right| = \lceil zKd \rceil$ и следователно $a = \left(-1\right)^{m/2+1}\lceil zKd \rceil$. Остава реалното пресмятане на *z*, което се свежда до това, при дадени $s > 1$ и $\varepsilon > 0$ да се намери положително цяло число *N*, за което в стъпка 4 на алгоритъма да е гарантирано, че $0 \leq \zeta\left(s\right) - z < \varepsilon$. Вече имаме $0 \leq \zeta\left(s\right) - z$, тъй като *z* е приближение отдолу. Освен това може да се провери (в произведението са прости числа), че $\sum_{n \leq N} n^{-s} \leq \prod_{p \leq N}\left(1 - p^{-s}\right)^{-1}$, следователно

(1.65) $$\zeta\left(s\right) - z \leq \sum_{n=N+1}^{\infty} n^{-s} \leq \int_N^{\infty} x^{-s}dx = \frac{1}{\left(s-1\right)N^{s-1}}.$$

Ако изберем $N > \varepsilon^{-1/(s-1)}$, то $\frac{1}{\left(s-1\right)N^{s-1}} \leq \frac{1}{N^{s-1}} < \varepsilon$, което води до $\zeta\left(s\right) - z < \varepsilon$. За нашите цели имаме $s = m$ и $\varepsilon = \left(Kd\right)^{-1}$. Следователно е достатъчно да изберем $N > \left(Kd\right)^{1/(m-1)}$.

Константата на Ойлер-Маскерони $\gamma = \lim_{n\to\infty}\left(\sum_{k=1}^{n}\frac{1}{k} - \ln(n)\right)$.

За константата на Ойлер γ е използван алгоритъм, предложен от Брент и МакМилан в [37].

Предполагаме, че желаем да пресметнем ойлеровата константа γ с точност от d десетични знака. Ако изберем n да е най-голямото цяло число, което е по-малко от $c + (1/4)\ln(10)d$ при подходяща константа c, то

$$\left|\gamma - \frac{U(n)}{V(n)}\right| < \pi e^{4-4c} 10^{-d}$$

където $U(n)$ и $V(n)$ се пресмятат както следва:

Определят се

$$A_k = \left(\frac{n^k}{k!}\right)\left(H_k - \ln(k)\right), \quad U_k = \sum_{j=0}^{k} A_j$$

$$B_k = \left(\frac{n^k}{k!}\right)^2, \quad V_k = \sum_{j=0}^{k} B_j$$

(Тук H_k е k-тото хармонично число (1 + ½ + … + 1/ k)). Тогава $A_0 = -\ln(n)$, $B_0 = 1$, $U_0 = A_0$, $V_0 = 1$ и за k = 1, 2, … получаваме

$$B_k = B_{k-1}\frac{n^2}{k^2}, \quad A_k = \frac{\left(A_{k-1}\frac{n^2}{k} + B_k\right)}{k},$$

$$U_k = U_{k-1} + A_k, \quad V_k = V_{k-1} + B_k.$$

Интегрален синус и косинус.

(1.66)
$$Si(x) = \int_0^x \frac{\sin(t)}{t}dt, \quad Ci(x) = -\int_x^\infty \frac{\cos(t)}{t}dt$$

Използвани са стандартни редове и асимптотични разложения.

Представяне с редове:

(1.67)
$$\begin{cases} Si(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!(2n+1)} \\ Ci(x) = \gamma + \ln(x) + \sum_{n=1}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!(2n)} \end{cases},$$

Асимптотичното представяне при големи стойности на x е

(1.68)
$$\begin{cases} Si(x) = \frac{\pi}{2} - f(x)\cos(x) - g(x)\sin(x), \\ Ci(x) = f(x)\sin(x) - g(x)\cos(x), \end{cases}$$

където

(1.69)
$$\begin{cases} f(x) \sim \frac{1}{x}\left(1-\frac{2!}{x^2}+\frac{4!}{x^4}-\frac{6!}{x^6}+\cdots\right), \\ g(x) \sim \frac{1}{x^2}\left(1-\frac{3!}{x^2}+\frac{5!}{x^4}-\frac{7!}{x^6}+\cdots\right). \end{cases}$$

Интегрална експонента (реален аргумент).

(1.70)
$$\mathrm{Ei}(x) = \int_{-\infty}^{x} \frac{e^t}{t} dt, x \neq 0.$$

Представянето със сходящ ред е

(1.71)
$$\mathrm{Ei}(x) = \gamma + \ln|x| + \sum_{n=1}^{\infty} \frac{x^n}{n!n}.$$

Асимтотическото представяне при $x \to +\infty$ е

(1.72)
$$\mathrm{Ei}(x) \sim \frac{e^x}{x}\left(1+\frac{1!}{x}+\frac{2!}{x^2}+\frac{3!}{x^3}+\cdots\right).$$

Интегрален логаритъм (реален аргумент).

(1.73)
$$\mathrm{li}(x) = \int_0^x \frac{dt}{\ln(t)}, x \neq 1.$$

За пресмятане се използва зависимостта

(1.74)
$$\mathrm{li}(x) = \mathrm{Ei}\left(\ln\left(x\right)\right).$$

.

Елиптични функции на Якоби ([106],[152]).

Елиптичните функции на Якоби могат да се определят чрез тета функциите, което ни дава възможност за пряко изчисление с тях.

Нека $\mathrm{K}(k)$ е стойността е пълния елиптичен интеграл от първи род за аргумент *k.* Полагаме $k' = \sqrt{1-k^2}$ и $\mathrm{K}'(k) = \mathrm{K}(k') = \mathrm{K}\left(\sqrt{1-k^2}\right)$.

Определяме $q = \exp\left(-\pi\, \mathrm{K}'(k) / \mathrm{K}(k)\right)$ и имаме

(1.75)
$$k = \frac{\theta_2^2(0,q)}{\theta_3^3(0,q)}, \quad k' = \frac{\theta_4^2(0,q)}{\theta_3^3(0,q)}, \quad \mathrm{K}(k) = \frac{\pi}{2}\theta_3^2(0,q).$$

Като положим $\zeta = \frac{\pi z}{2\,\mathrm{K}(k)}$,

(1.76) $$\mathrm{sn}(z,k)=\frac{1}{\sqrt{k}}\frac{\theta_1(\zeta,q)}{\theta_4(\zeta,q)}=\frac{1}{\mathrm{ns}(z,k)},$$

(1.77) $$\mathrm{cn}(z,k)=\sqrt{\frac{k'}{k}}\frac{\theta_2(\zeta,q)}{\theta_4(\zeta,q)}=\frac{1}{\mathrm{nc}(z,k)},$$

(1.78) $$\mathrm{dn}(z,k)=\sqrt{k'}\frac{\theta_3(\zeta,q)}{\theta_4(\zeta,q)}=\frac{1}{\mathrm{nd}(z,k)}.$$

Останалите функции са съответно

(1.79) $$\mathrm{sd}(z,k)=\frac{\mathrm{sn}(z,k)}{\mathrm{dn}(z,k)}=\frac{1}{\mathrm{ds}(z,k)},$$

(1.80) $$\mathrm{cd}(z,k)=\frac{\mathrm{cn}(z,k)}{\mathrm{dn}(z,k)}=\frac{1}{\mathrm{dc}(z,k)},$$

(1.81) $$\mathrm{sc}(z,k)=\frac{\mathrm{sn}(z,k)}{\mathrm{cn}(z,k)}=\frac{1}{\mathrm{cs}(z,k)}.$$

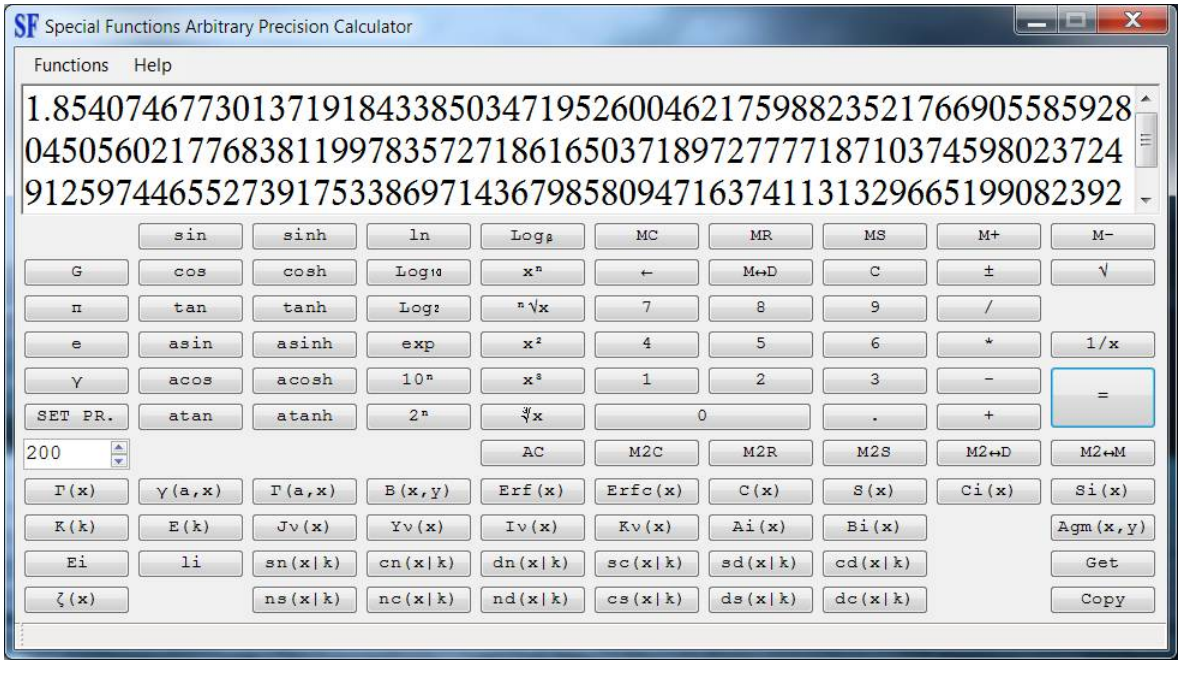
1.2. Интегриране на реализираните методи за пресмятане на елементарни и специални функции в отделен спомагателен програмен инструмент SFCALC.

Процесът на тестване е една от най-трудоемките операции по съставяне на библиотеката с елементарни и специални функции. Основните проверки са за предварително известни стойности на дадена функция при точно определен аргумент (аргументи). Още по-добра проверка е възможността за сравняване на даден резултат, когато той може да се изрази чрез различни функции и следователно с използване на различни методи. Добавянето на още функции спомага за този тип проверка. В мрежата има много онлайн калкулатори за различни видове специални функции, които също са удобни за първоначална настройка. Повечето от тях обаче са с ограничена точност. Естествена възможност е успоредното ползване на друга програмна среда (например използваща mpmath в Python). Удобно е обаче да има на разположение и интерактивна програма за тестване. С цел удобство проверката на функции е реализиран прототип на калкулатор. Калкулаторът позволява динамична промяна на използваната точност. Полето на дисплея автоматично добавя вертикален плъзгач за превъртане при нужда, ако текущата точност на пресмятанията го изисква. Форматът на показваните числа се мени автоматично според това дали резултатът се нуждае от експоненциален формат. Извършва се и подходящо закръгляване. Има индикация за очакване на аргумент при съответна функция с повече от един аргумент. Цифрите, десетичната точка и аритметичните операции могат да се въвеждат и от клавиатурата.

Един пример за пресмятане с калкулатора при точност 200 знака:

(1.82) $$K\left(\frac{1}{\sqrt{2}}\right)=\frac{1}{4\sqrt{\pi}}\Gamma\left(\frac{1}{4}\right)^2$$

Последователното набиране на: $4,1/x,\Gamma(x),x^2,\mathrm{MS},\pi,\sqrt{\ },1/x,/,4,=,*,\mathrm{MR},=,$ дава дясната част на равенството.



Фигура 1.3. Пресмятане на $\frac{1}{4\sqrt{\pi}}\Gamma\left(\frac{1}{4}\right)^2$ с 200 десетични знака в SFCALC.

Може да се запомни с MS и след набиране на 2, $\sqrt{\ }$, 1/x, K(x) се получава лявата страна в (79) - пълен елиптичен интеграл от първи род за стойност $1/\sqrt{2}$. Не е приведена илюстрация, защото резултатът е един и същ.

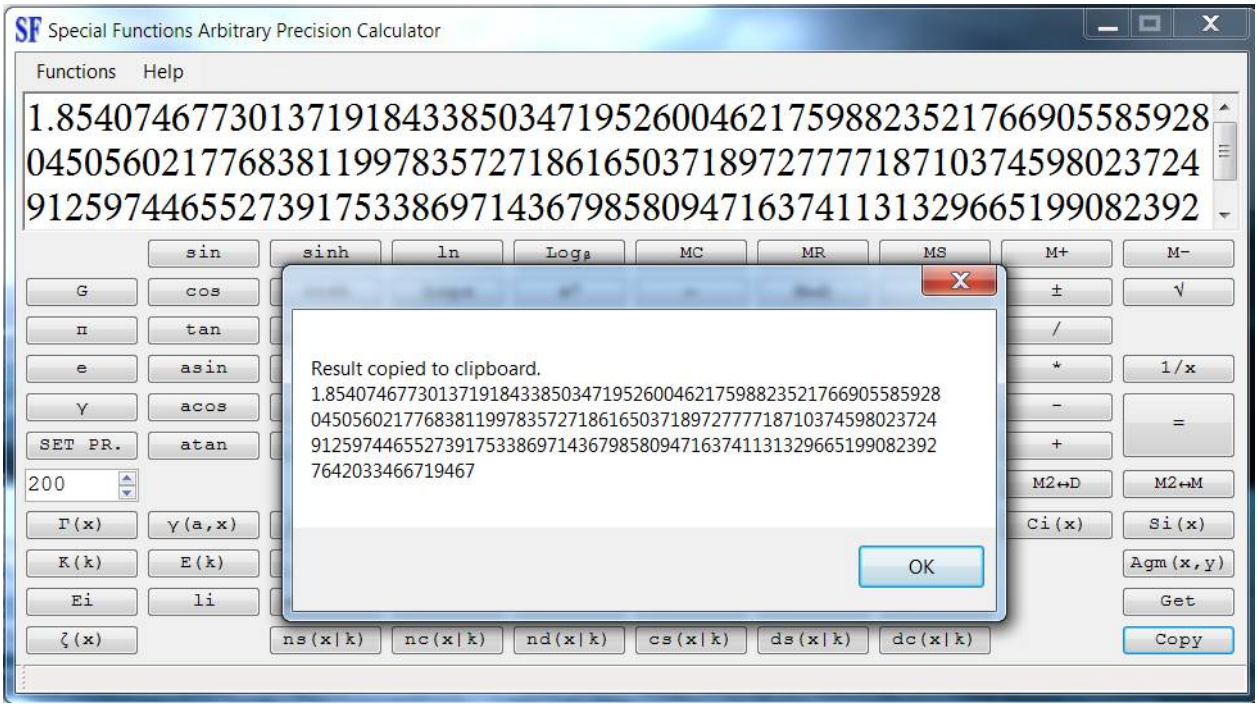
За улесняване на тестването в случаи, когато еквивалентният израз е по-сложен, са добавени допълнителни възможности: запаметяване във втори регистър (M2S) и $M \leftrightarrow D$ и $M2 \leftrightarrow D$, което разменя местата на последния получен и последния запаметен в съответния помощен регистър резултат.

Приведената по-горе илюстрация на фиг. 1.3 е типична в смисъл, че резултатът от пресмятането не се вижда изцяло и изисква превъртане на съответното поле, за да се види изцяло, в случая,

1.85407467730137191843385034719526004621759882352176690558592804505602
17768381199783572718616503718972777718710374598023724912597446552739175
53386971436798580947163741131329665199082392764203346671946 7.

Освен това обаче се оказа, че в много случаи калкулаторът е изключително полезен като осигуряващ параметри при формулиране на задача, решавана от друго програмен инструмент на система, както е описано в съответните глави по-нататък. За целта бе предвидена възможност за копиране на резултат в клипборда с цел бъдещо използване на

друго място. Това става с бутона "Copy" (има и възможност за внасяне на резултата от клипборда с "Get").



Фигура 1.4. Копиране на резултат от SFCALC в клипборда.

В първоначалния замисъл на изследването за реализацията на „калкулатора” се предвиждаше той да се превърне в справочник за съответните специални функции (в Help). Идеята обаче изисква доста време за реализация и за момента е отложена.

1.3. Заключение.

Осъществена е програмната реализация в целевата програмна среда за пресмятане на елементарни и специални математически функции, използвани при разработването на средствата за решаване на различни задачи на числения анализ, някои от които са описани в следващите глави. На базата на предложената реализация е разработено самостоятелен програмен инструмент SFCALC, даващ следните предимства: 1: облекчава възможността за тестване и 2: позволява удобно интерактивно генериране на начални условия, изисквани от други програмни инструменти в разработената система.

Глава 2. Методи и средства за подобряване численото пресмятане на някои класове определени интеграли с висока точност.

2.1. Използване на двойно експоненциално преобразование за числено пресмятане на определени интеграли.

2.1.1. Предварителни сведения. Формула на Ойлер-Маклорен и преобразованието tanh-sinh.

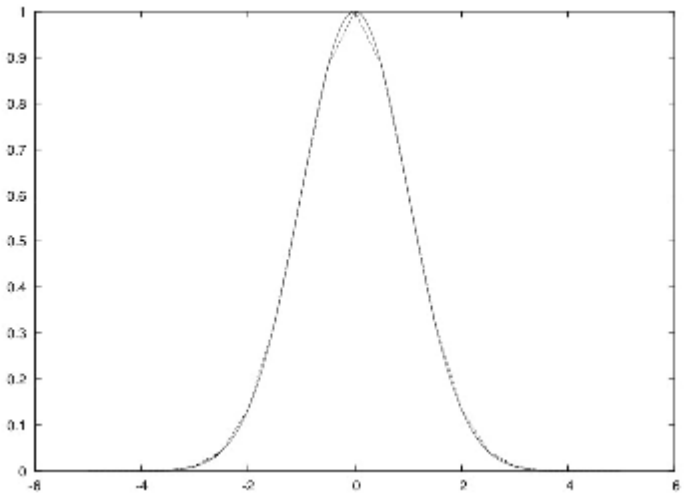
За да се обоснове използваният метод, се изисква известно предварително обяснение. Нека е даден краен интервал (a,b), както и положителни цели числа m и n. Полагаме $h=(b-a)/n$ и $h_j=a+jh$ (а B_{2i} бележат числа на Бернули). От формулата на Ойлер-Маклорен (предполагайки че функцията е поне $2m+2$ пъти непрекъснато диференцируема)

$$\begin{aligned}\int_a^b f(x)\mathrm{d}x &= h\sum_{j=0}^{n} f(x_j)-\frac{h}{2}(f(a)+f(b)) \\ &-\sum_{i=1}^{m}\frac{h^{2i}B_{2i}}{(2i)!}(f^{(2i-1)}(b)-f^{(2i-1)}(a)) \\ &-E(h,m),\end{aligned} \tag{2.1}$$

където

$$E(h,m)=\frac{h^{2m+2}}{(2m+2)!}(b-a)B_{2m+2}f^{(2m+2)}(\xi) \tag{2.2}$$

за някакво $\xi\in(a,b)$, се вижда, че ако функцията f има производни от произволен ред и всичките и производни са със стойност нула в a и b (като в камбановидна крива), членът на грешката $E(h,m)$ се стреми към нула по-бързо от всяка степен на h.



Фигура 2.1. Правило на трапеците за камбановидна крива.

Формулата на Ойлер-Маклорен може да се разглежда, като предоставяща корекция от по-висок ред за правилото на трапеците.

Нека е дадена функция f, определена в $(-1,1)$. Да положим $g(t)=\tanh\left(\frac{\pi}{2}\sinh(t)\right)$.

След замяната $x=g(t)$ получаваме

$$(2.3)\quad \boxed{I=\int_{-1}^{1} f(x)\,\mathrm{d}x=\int_{-\infty}^{\infty} f\left(g(t)\right)g'(t)\,\mathrm{d}t\approx h\sum_{j=-N}^{N} w_j f(x_j)=I_h^N,\quad x_j=g\left(hj\right),\ w_j=g'\left(hj\right)}.$$

Това е всъщност квадратурната формула tanh-sinh. Тъй като $g'(t)$ клони към нула много бързо с увеличаване на $|t|$, произведението $f\left(g(t)\right)g'(t)$ е обикновено функция с добре изразена камбановидна форма. Както бе отбелязано в предишния абзац, за такива функции формулата на Ойлер-Маклорен гарантира, че сумата по-горе е изключително точно приближение. Обикновено двукратното намаляване на h удвоява броя на точните цифри. Ако функцията е аналитична в отворения интервал и положим $I_h=I_h^\infty=h\sum_{j=-\infty}^{\infty} w_j f(x_j)$, то $\left|I-I_h\right|\approx\exp\left(-\frac{c_1}{h}\right)$ и $\left|I-I_h^N\right|\approx\exp\left(-c_2\frac{N}{\ln N}\right)$.

2.1.2. Вариации на двойно експоненциалната формула.

Приведеното в предишния раздел преобразование tanh-sinh е за случая на крайния интервал $(-1,1)$. Обобщението за произволен краен интервал (a,b) е очевидно, чрез линейно преобразование

$$(2.4)\quad \int_a^b f(t)\,\mathrm{d}t=\frac{b-a}{2}\int_{-1}^{1} f\left(\frac{a+b}{2}+\frac{b-a}{2}x\right)\mathrm{d}x\,.$$

В таблицата по-долу са приведени подобни преобразования и за други случаи.

Тип на интеграла	Преобразование
$\int_{-1}^{1} f(x)\,\mathrm{d}x$	$x = \tanh\left(\frac{\pi}{2}\sinh(t)\right)$
$\int_{0}^{\infty} f(x)\,\mathrm{d}x$	$x = \exp\left(\frac{\pi}{2}\sinh(t)\right)$
$\int_{-\infty}^{\infty} f(x)\,\mathrm{d}x$	$x = \sinh\left(\frac{\pi}{2}\sinh(t)\right)$
$\int_{0}^{\infty} f_1(x)e^{-x}\,\mathrm{d}x$	$x = \exp\left(t - \exp(-t)\right)$

Таблица 2.1. Двойно експоненциално преобразование за различни типове интеграли.

Схемите за пресмятане, които се получават с тях, имат сходни свойства. Обаче абсцисите и теглата са различни и трябва да се пресмятат отделно. Често това допълнително пресмятане може да се избегне, като се използва например равенството

$$(2.5)\quad \int_{0}^{\infty} f(t)\,\mathrm{d}t = \int_{0}^{1}\left(f(x) + f(1/x)/x^2\right)\mathrm{d}x\,.$$

2.1.3. Алгоритъм за формулата tanh-sinh.

Абсцисите и теглата не зависят от функцията и са разпределени равномерно със стъпка h. Това обстоятелство дава възможност за ефективна реализация на алгоритъм, използващ няколко нива на разполовяване на стъпката. Абсцисите и теглата се пресмятат за минимална, предварително зададена стъпка ($h = 2^{-m}$ за някакво "ниво" m), след което се обхождат по нива (за $n = m - k < m$) и се пресмята сумата за всяко ниво чрез пресмятане на функцията за точки със стъпка $1/2^{m-k}$. Така пресметната сума за дадено ниво се използва след това при пресмятане на следващото ниво на разполовяване. Ако последните две пресметнати суми се различават по-малко от желаната точност, процесът приключва. При такава организация, разбира се, трябва да се предоставят достатъчен брой нива, т.е. такава минимална стъпка (на максималното ниво m), при която да се достига желаната точност. 12 нива са достатъчни за повечето интеграли за точност от порядъка на 1000 знака в повечето случаи [15].

По-долу е приведен алгоритъмът във вид на псевдокод [29], алгоритъмът е описан и в [147].

Вход: брой на нивата m, функция f.
Изход: приближението на интеграла hS
Инициализация:
$h := 2^{-m}$
for k := 0 to $20 \cdot 2^{m}$ **do**
  t := kh

$x_k := \tanh(\pi/2 \cdot \sinh(t))$
$w_k := \pi/2 \cdot \cosh(t) / \cosh^2(\pi/2 \cdot \sinh(t))$
if $|x_k - 1| < \varepsilon$ **then**
exit **do**
end if
end for
$n_t := k$ (стойността на k при излизане)

Квадратура:
$S := 0$, $h := 1$
for $k := 1$ to m (или до достигане на желаната точност) **do**
$h := h / 2$
for $i := 0$ to n_t step 2^{m-k} **do**
if mod(i, 2^{m-k+1}) $\neq 0$ or $k = 1$ **then**
if $i = 0$ **then**
$S := S + w_0 f(0)$
else
$S := S + w_i \left(f(-x_i) + f(x_i) \right)$
end if
end if
end for
end for
Result = hS

Фигура 2.2. Основна схема на алгоритъма tanh-sinh.

Тази основна схема позволява някои вариации, в които например условието за завършване при инициализацията не е $|x_k - 1| < \varepsilon$, а $w_k < \varepsilon^2$ и в същото време пресмятанията се извършват с точност ε^2 [15].

Преобразованията, описани по-горе, не са подходящи за интеграли от типа

$$(2.6)\ I_s = \int_0^\infty f_1(x)\sin(\omega x)\,\mathrm{d}x,$$

$$(2.7)\ I_c = \int_0^\infty f_1(x)\cos(\omega x)\,\mathrm{d}x$$

и за някои подобни, включващи бавно намаляващи осцилиращи функции, например функции на Бесел. [17] дава съвременно изложение на задачата за пресмятане с висока точност на определени интеграли с осцилиращи функции. В [107] е предложено преобразование на променливата, което е подходящо за интеграли от този тип. Реализацията на специални средства за решаване на подобен тип интеграли в настоящата програмна система предстои.

2.2. Реализация на вариация на метода на двойната експонента в отделен графичен програмен инструмент NQTS.

2.2.1. Предназначение и възможности.

Програмният инструмент NQTS е предназначен за числено пресмятане на определени интеграли с висока точност. Областта на интегриране може да бъде краен или безкраен интервал. Безкрайният интервал може да е от типа (a,∞) с произволно реално число a или $(-\infty,\infty)$. Изчисленията се извършват с произволна (задавана от потребителя) точност. Изразът за подинтегралната функция се задава в текстова форма и може да включва алгебрични операции и функции (елементарни и специални, вж. списъка в раздела " Помощни средства за изграждане на програмни инструменти" на увода). В израза се допускат допълнителни параметри. Т.е. интерпретаторът приема произволен брой променливи, но тези от тях, които са различни от основната променлива на задачата (x) се считат за параметри, за чиято инициализация се изисква отделно задаване от потребителя.

Потребителят може да промени желаната грешка за резултата 10^{-p}, като въведе и потвърди съответното цяло число p в текстовото поле под бутона “Error (desired)”. В реализираната схема точността на пресмятанията (броят на значещите десетични знаци за междинните пресмятания) е винаги равна на $2p$. След промяна на желаната грешка за резултата, точността автоматично се променя на $2p$. Дадена е възможност за промяна на точността на пресмятанията чрез въвеждане и потвърждаване на съответното цяло число в текстовото поле под бутона “Precision”. По същия начин, след като тази точност се промени, желаната грешка за резултата се фиксира автоматично на половината на точността. Понастоящем наличието на два начина за промяна на точността изглежда като нещо излишно, но е запазена за бъдеща употреба в по-общи условия и е полезна за напомняне на текущата точност на пресмятанията. Тази инициализация не зависи от функцията и е реализирана в асинхронен процес, така че потребителят може да започне да въвежда задача, без да е необходимо да изчаква завършването на инициализацията. Ако все пак въвеждането завърши преди това и потребителят щракне с мишката върху бутона Compute, появява се съобщение, което уведомява потребителя за ситуацията (Initialization not yet finished, please wait).

Интерфейсът отчита трите типа възможни задачи.

- *Краен интервал.*

В този случай полетата за избор на тип безкраен интервал са изчистени. Полетата за границите на интервала (a and b) стават достъпни.

- *Безкраен интервал от типа* (a,∞).

В този случай в съответното поле за избор е поставена отметка. Евентуалната отметка в другото поле за избор се премахва автоматично. Полето за дясна граница на интервала (b) става недостъпно.

- *Цялата реална права - интервал от типа* $(-\infty,\infty)$.

В този случай в съответното поле за избор е поставена отметка. Евентуалната отметка в другото поле за избор се премахва автоматично. И двете полета за граници на интервала (a и b) стават недостъпни.

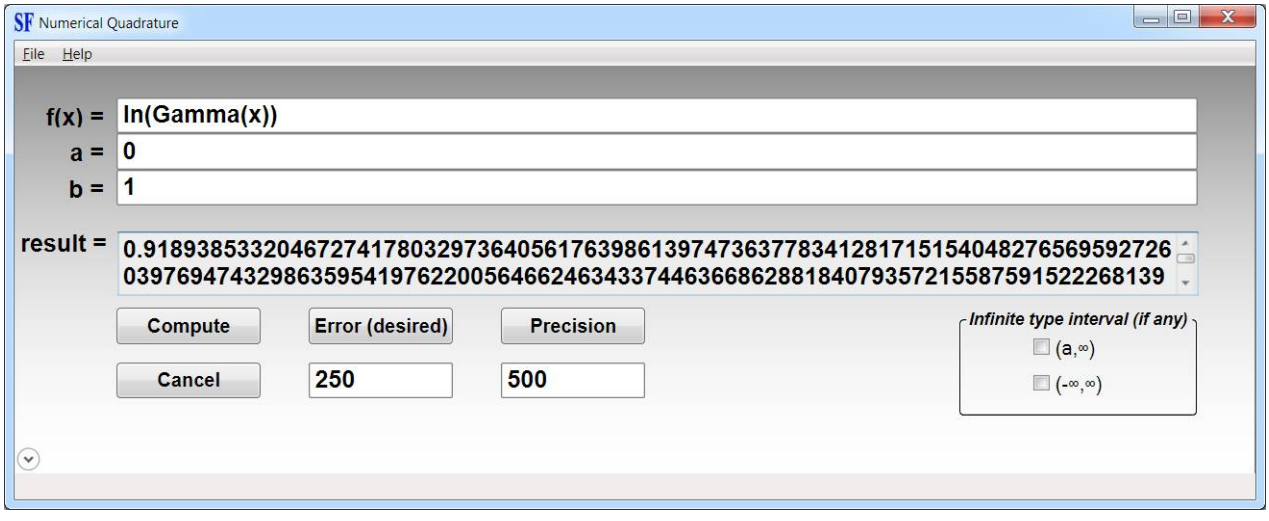
Докато се изчислява резултатът, в лентата на състоянието е изведено съобщението

“Computing…”.

2.2.2. Някои примери.

Програмният инструмент е тестван с различни примери за умерена точност (няколко стотин и дори хиляди знака), включително за списъка от 14 примерни интеграла в [18]. Ето някои примери, с прости граници и без параметри.

С първоначалната точност (500 десетични знака, грешка на резултата 10^{-250}) за $\int_0^1 \ln(\Gamma(x))\mathrm{d}x$ получаваме



Фигура 2.3. Резултат от числено решаване на интеграла $\int_0^1 \ln(\Gamma(x))\mathrm{d}x$ с NQTS.

В полето на резултата (трябва да се превърти, за да се видят всички цифри) имаме

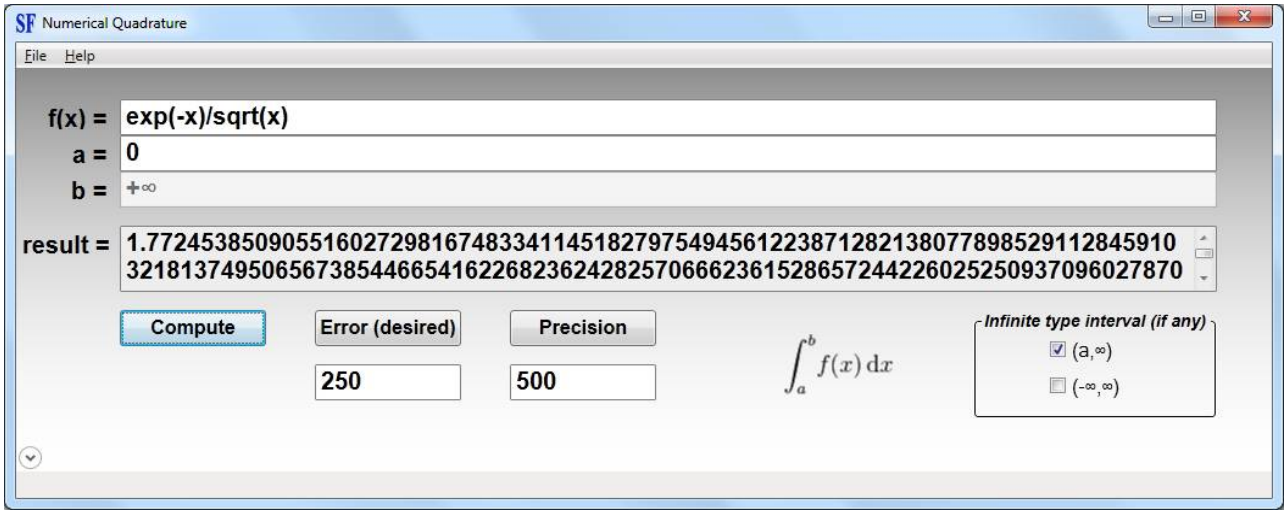
0.91893853320467274178032973640561763986139747363778341281715154048276
56959272603976947432986359541976220056466246343374463668628818407935721
55875915222681393603560742547358669046395905991380805630163234873094627
37462551825169495447741009585935139198**16**.

Точният резултат е $\frac{\ln(2\pi)}{2}$ и може да се пресметне лесно с SFCALC. Полученият с SFCALC резултат (точност 251 десетични знака, NQTS дава един знак повече в резултата) е

0.91893853320467274178032973640561763986139747363778341281715154048276
56959272603976947432986359541976220056466246343374463668628818407935721
55875915222681393603560742547358669046395905991380805630163234873094627
37462551825169495447741009585935139198**15**.

Разликата е в последния знак.

За $\int_0^{\infty} \frac{e^{-t}}{\sqrt{t}} \mathrm{d}t$ с точност 10^{-250} се получава



Фигура 2.4. Резултат от численото решаване на интеграла $\int_0^{\infty} \frac{e^{-t}}{\sqrt{t}} \mathrm{d}t$ с NQTS.

В полето на резултата (трябва да се превърти, за да се видят всички цифри) имаме

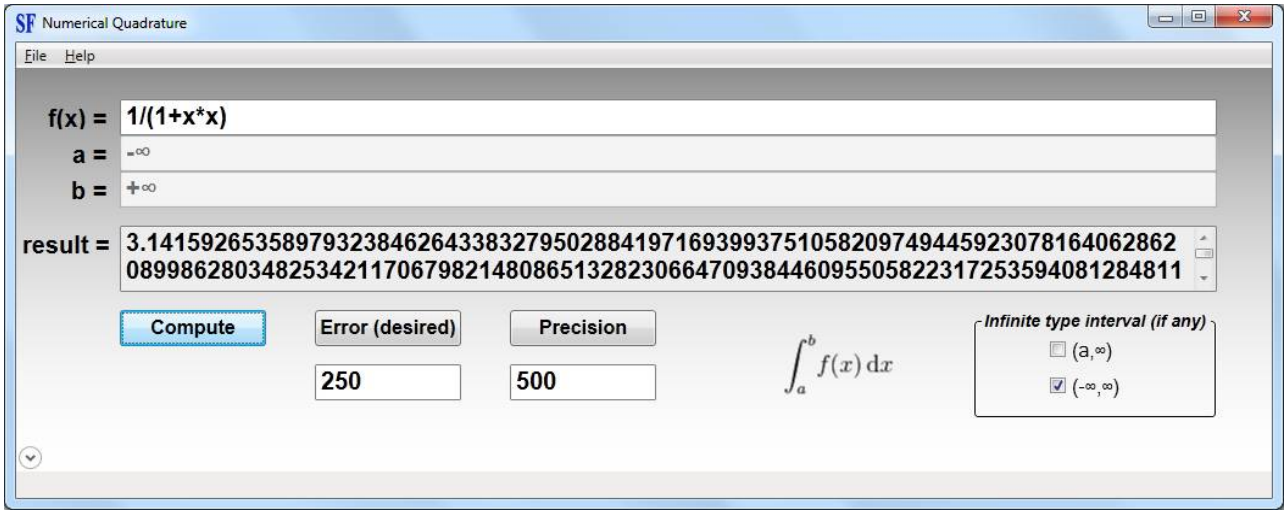
1.772453850905516027298167483341145182797549456122387128213807789852911284591032181374950656738544665416226823624282570666236152865724422602525093709602787068462037698653105122849925173028950826228932095379267962800174639015351479720516700190185234019.

Точният резултат е $\sqrt{\pi}$ и SFCALC с точност от 250 десетични знака дава

1.772453850905516027298167483341145182797549456122387128213807789852911284591032181374950656738544665416226823624282570666236152865724422602525093709602787068462037698653105122849925173028950826228932095379267962800174639015351479720516700190185234002,

което е същото с гореспоменатата особеност - последните два знака са закръглени правилно спрямо резултата от NQTS, който дава всъщност 251 знака.

Константата π може да се пресметне като стойността на интеграла $\int_{-\infty}^{\infty} \frac{1}{1+x^2} \mathrm{d}x$.



Фигура 2.5. Резултат от численото решаване на интеграла $\int_{-\infty}^{\infty} \frac{1}{1+x^2}\mathrm{dx}$ с NQTS.

В полето на резултата (трябва да се превърти, за да се видят всички десетични знаци) имаме

3.14159265358979323846264338327950288419716939937510582097494459230781
64062862089986280348253421170679821480865132823066470938446095505822317
25359408128481117450284102701938521105559644622948954930381964428810975
66593344612847564823378678316527120190**9**.

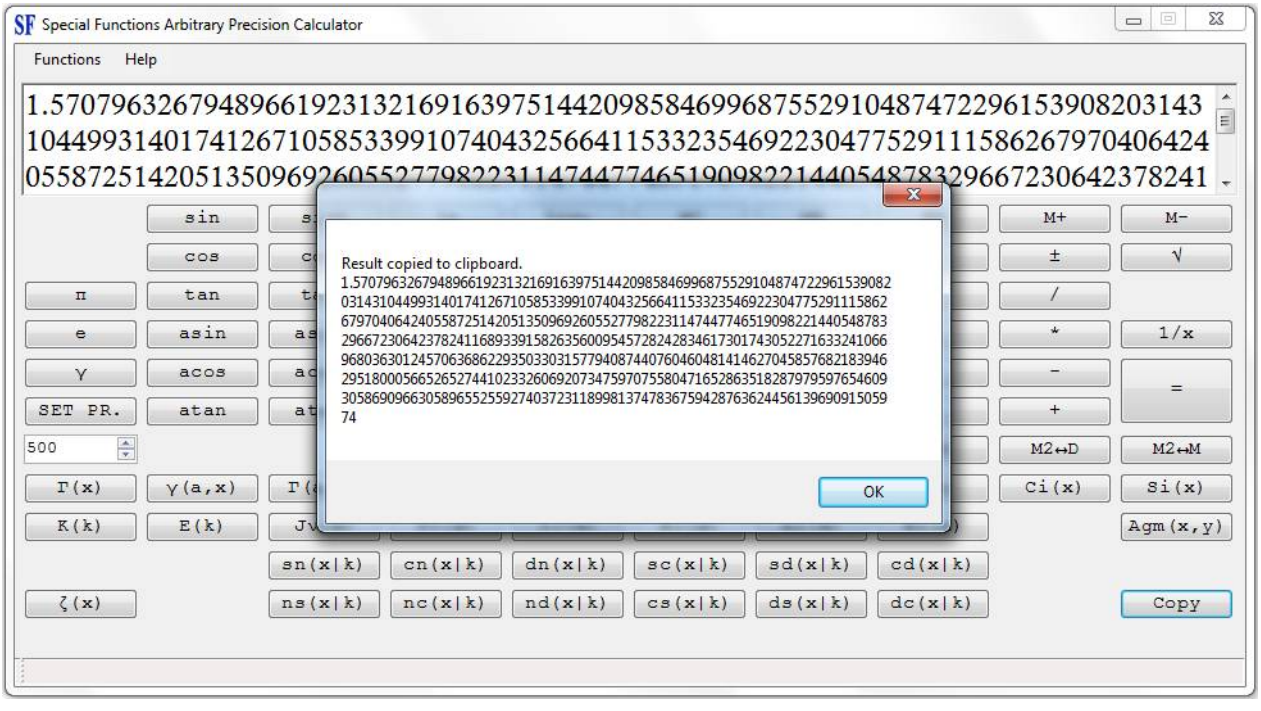
SFCALC дава за π (с точност 250 десетични знака)

3.14159 ... 12019**1,**

един знак по-малко, правилно закръглен.

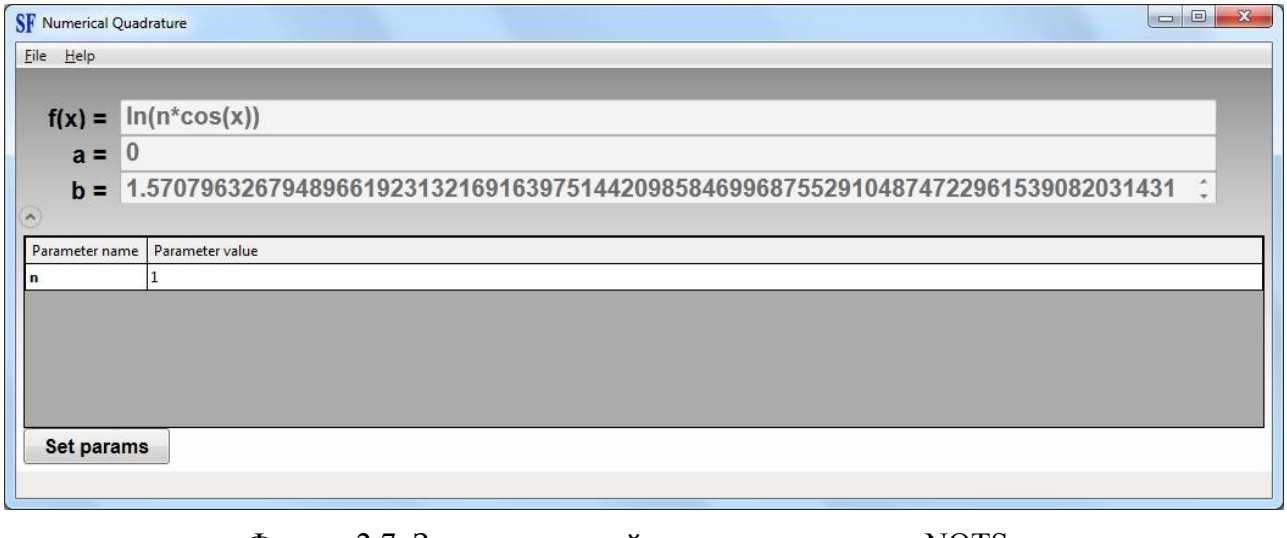
2.2.3. Задаване на параметри и гранични условия с необходимата точност с помощта на SFCALC.

Ако за инициализирането на някакво поле или параметър е нужна определена математическа константа, може да се използва програмният инструмент SFCALC. Бутонът Copy на SFCALC прехвърля изчисления резултат в клипборда. От там може да се копира с натискане на Ctrl+V, след изчистване на съответното поле за граница на интервала или за параметър. В израза за функцията могат да се използват директно π, e, γ с имената pi, e, gamma, но понякога присъстват конкретни константи, които по принцип би могло да се зададат и като резултат от функции, приложени върху константа, например да се запише sqrt(2), но това би се изчислявало интерпретатора многократно. Задаването на параметър решава въпроса. А при задаването на границите на интегриране параметри няма, така че там задължително трябва да се задава конкретна стойност. Като илюстрация може да се използва $\int_0^{\frac{\pi}{2}} \ln(n\cos(x))\,\mathrm{d}x$, където n е параметър, който ще зададем като 1. С SFCALC се пресмята $\pi/2$ с необходимата точност: въвежда се (и се потвърждава) 500 под бутона "SET PR.". След това се избират последователно бутоните π, /, 2, = и накрая с бутона Copy, се прехвърля константата $\pi/2$ в клипборда като текст.



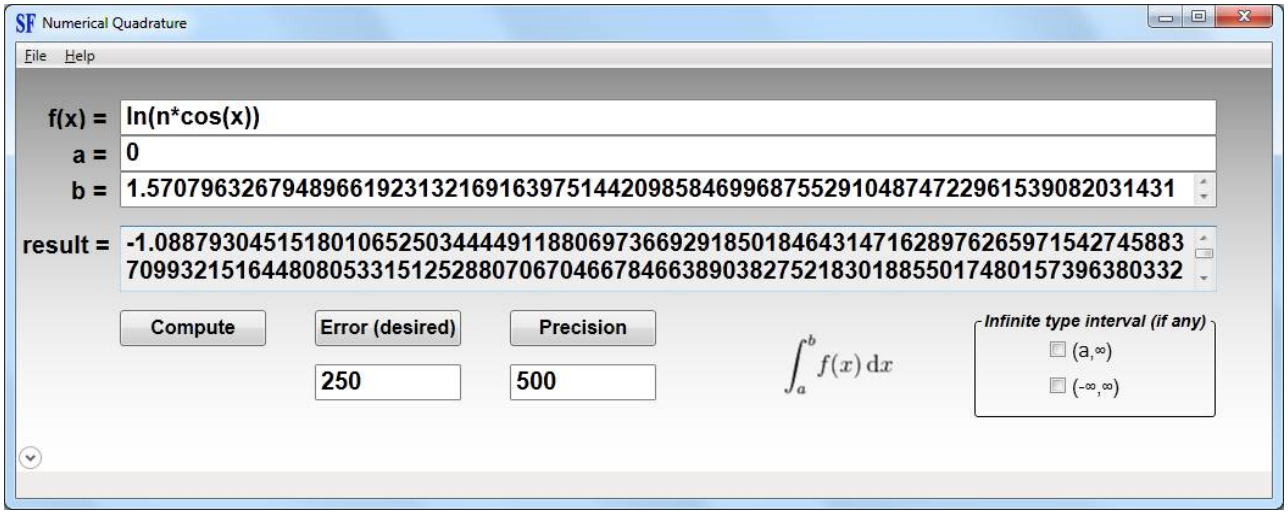
Фигура 2.6. Прехвърляне на желания резултат от SFCALC в клипборда.

Изчиства се полето за дясна граница на интервала (b) и се копира (Ctrl+V). Това прехвърля резултата в полето (вижда се краят – трябва да се превърти нагоре за да се стигне до началото). В полето за левия край на интервала се въвежда 0, ако е нужно. Въвежда се $\ln(n*\cos(x))$ в полето за функцията и пресмятането започва с бутона Compute. Това отваря таблица за параметри. За n се въвежда 1 и се потвърждава с "Set params".



Фигура 2.7. Задаване на стойност на параметър в NQTS.

След пресмятането се получава



Фигура 2.8. Резултат от численото пресмятане на интеграла $\int_0^{\frac{\pi}{2}} \ln(\cos(x))\,\mathrm{d}x$ в NQTS.

Полето на резултата съдържа

1.088793045151801065250344449118806973669291850184643147162897626597154274588370993215164480805331512528807067046678466389038275218301885501748015739638033208354306398902900085594241828542310963591445370660726023127015955425916345915993755144714047**288**.

с отрицателен знак. Точният резултат е $-\pi\ln(2)/2$. Пресмятането на константата (без знака минус) с SFCALC (точност 250 десетични знака) дава

1.088793045...404**729**.

И накрая един интересен пример от [12], чиято стойност е известна, но затруднява специализирани среди за математически пресмятания с отгатването на аналитичната (затворена) форма на резултата.

$$\int_0^{\pi/2} \frac{\arcsin\left(\sqrt{2}/2\cdot\sin x\right)\sin x}{\sqrt{4-2\sin^2 x}}\,\mathrm{d}x = \frac{\sqrt{2}\pi\ln 2}{8}. \tag{2.8}$$

Доста продължително пресмятане на лявата част с NQTS дава (за 1000 десетични знака при работна точност 2000 знака): 0.3849464727 ... 3093521683.

Пресмятането на дясната част в (20) с SFCALC при точност 1002 десетични знака дава: 0.3849464727 ... 3093521683**2**.

Реализацията в NQTS на схема на преобразование от типа в [107], позволяваща пресмятане на интеграли от бавно намаляващи осцилиращи функции все още предстои.

2.3. Квадратурна формула на Clenshaw-Curtis.

2.3.1. Някои предварителни сведения.

За да се обясни ефективността на тази конкретна квадратурна формула за гладки периодични функции, трябва да се напомнят някои понятия и факти, свързани с преобразованията на Фурие. Да предположим, че функцията f е определена в $\mathbb{R}$. Преобразованието на Фурие за f се определя с $\tilde{f}(\omega)=\frac{1}{2\pi}\int_{-\infty}^{\infty} f(t)e^{-i\omega t}\,\mathrm{dt}$. С обратното преобразование получаваме оригинала $f(t)=\int_{-\infty}^{\infty}\tilde{f}(\omega)e^{i\omega t}\,\mathrm{d}\omega$. Определянето на класа от функции, за които това е възможно (по-широк от класа $L^1(\mathbb{R})$, т.е. функции, за които $\int_{-\infty}^{\infty}|f(t)|\mathrm{dt}<\infty$), е доста сложно и няма да навлизаме в него. За нас е важна идеята, свързана с връзката между гладкостта на f и бързината на намаляване на $\tilde{f}$. Тази връзка се формулира обикновено като теорема от тип Paley-Wiener: Ако $f(t)$ и първите ѝ n-1 производни са непрекъснати, а n-тата производна е с ограничена вариация, то при $\omega\to\infty$ $\tilde{f}(\omega)$ намалява поне толкова бързо, колкото и $|\omega|^{-(n+1)}$. В случай на периодични функции, ако честотата ω е кратна на основната честота $\omega_0=\frac{2\pi}{T}$, то $\tilde{f}(\omega)=\infty$, в противен случай $\tilde{f}(\omega)=0$. Формално в този случай имаме $\tilde{f}(\omega)=\sum_{\nu=-\infty}^{\infty}\tilde{f}_\nu\delta(\omega-\nu\omega_0)$. Другият начин на записване (в този случай на периодична функция с период T) е $f(t)=\sum_{\nu=-\infty}^{\infty}\tilde{f}_\nu e^{i\nu\omega_0 t}$, където $\tilde{f}_\nu=\frac{1}{T}\int_0^T f(t)e^{-i\nu\omega_0 t}\mathrm{dt}$.

Да предположим, че искаме да пресметнем интеграла $I=\int_0^T f(t)\,\mathrm{dt}$ с правилото на трапеците за N точки: $I_N^{tr}=\frac{T}{N}\left\{\frac{1}{2}[f(0)+f(T)]+\sum_{n=1}^{N-1}f\left(\frac{nT}{N}\right)\right\}$. Интегралът, който пресмятаме е всъщност $I=T\tilde{f}_0$. Правилото на трапеците ни дава $I_N^{tr}=T\tilde{f}_0+T\sum_{\nu\neq 0}\tilde{f}_{\nu N}$. В резултат на това получаваме оценка за грешката при прилагане на правилото на трапеците $E_N^{tr}=T\left|\sum_{\substack{\nu=-\infty\\ \nu\neq 0}}^{\infty}\tilde{f}_{\nu N}\right|$. Ако се предположи, че функцията е периодична (това включва $f^{(n)}(0)=f^{(n)}(T)$, n = 0, 1, 2, ...) и гладка (има непрекъснати производни от произволен ред), то според цитирания в предишния абзац резултат, величината на коефициентите $|\tilde{f}_\nu|$

в реда на Фурие намалява по-бързо от $1/P$, където P произволен полином от ν. В този случай доминиращият член в сумата за грешката е при $\nu = 1$, с типично поведение от вида на $\left|\tilde{f}_N\right| \sim e^{-\alpha|N|}$, за някаква константа α. Т.е. $E_N^{tr} \sim Te^{-\alpha|N|}$.

2.3.2. Квадратура на Clenshaw-Curtis.

За да се възползваме от бързата сходимост на правилото на трапеците за периодична (и гладка) функция, трябва да можем сведем задачата до интегриране на периодична функция в интервал равен на периода. По-надолу ще разглеждаме случая на интервала [-1,1], за който това става най-просто. В общия случай на краен интервал [a,b], може да се използва преобразованието $x = a + \frac{b-a}{2}(t+1)$. Разглеждаме интеграла

$$I = \int_{-1}^{1} f(t)\,\mathrm{dt}. \tag{2.9}$$

Замяната $t = \cos\theta$ ни дава

$$I = \int_{-1}^{1} f(t)\,\mathrm{dt} = \int_{0}^{\pi} f(\cos\theta)\sin\theta\,\mathrm{d\theta} = \int_{0}^{\pi} g(\theta)\sin\theta\,\mathrm{d\theta}, \tag{2.10}$$

където подинтегралната функция е периодична, но интегралът е върху половин период. Да разгледаме $g(\theta) = f(\cos\theta)$. За нея имаме $g(-\theta) = g(\theta)$, така че в нейния ред на Фурие присъстват само косинуси:

$$g(\theta) = \frac{\tilde{a}_0}{2} + \sum_{\nu=1}^{\infty} \tilde{a}_\nu \cos(\nu\theta) \text{ и} \tag{2.11}$$

$$\tilde{a}_\nu = \frac{2}{\pi}\int_0^{\pi} g(\theta)\cos(\nu\theta)\,\mathrm{d}\theta = \frac{1}{\pi}\int_0^{2\pi} g(\theta)\cos(\nu\theta)\,\mathrm{d}\theta. \tag{2.12}$$

Заместването на това представяне в последното равенство на (2.10) ни дава

$$I = \frac{\tilde{a}_0}{2}\int_0^{\pi}\sin\theta\,\mathrm{d}\theta + \sum_{\nu=1}^{\infty}\tilde{a}_\nu\left[\int_0^{\pi}\cos(\nu\theta)\sin\theta\,\mathrm{d}\theta\right]. \tag{2.13}$$

Тук интегралът в първи член е 2, а интегралите в сумата са нула за нечетно ν и $2/(1-\nu^2)$ за четно. Т.е. можем да запишем

$$I = \tilde{a}_0 + \sum_{\nu=1}^{\infty}\frac{2\tilde{a}_{2\nu}}{1-(2\nu)^2}. \tag{2.14}$$

Сега вече коефициентите се изразяват с интеграли от периодична функция за интервал равен на периода и можем да използваме за пресмятането им правилото на трапеците, разчитайки на изводите за бърза сходимост от предишния подраздел. По-конкретно, квадратурното правило за $N+1$ точки (четно N) е

(2.15) $$I \approx I_N^{CC} = \tilde{a}_0 + \sum_{\nu=1}^{N/2} \frac{2\tilde{a}_{2\nu}}{1-(2\nu)^2} = \tilde{a}_0 + \sum_{\substack{\nu=2 \\ \nu \text{ четно}}}^{N} \frac{2\tilde{a}_\nu}{1-\nu^2}.$$

Ако приемем, че **a** е вектор, чиито компоненти са $\tilde{a}_0, \tilde{a}_1, \cdots, \tilde{a}_N$, а **W** е векторът, определен с

(2.16) $$\boxed{W_\nu = \begin{cases} 1, \nu = 0 \\ \dfrac{2}{1-\nu^2}, \nu > 0, \textit{четно} \\ 0, \nu \textit{ нечетно} \end{cases}}$$

можем да запишем

(2.17) $$I_N^{CC} = \mathbf{W} \cdot \mathbf{a}.$$

Ако приложим правилото на трапеците с N точки за (2.12), получаваме

(2.18) $$\tilde{a}_\nu = \sum_n \Lambda_{\nu n} f(t_n),$$

където

(2.19) $$t_n = \cos\frac{n\pi}{N}, \quad \boxed{\Lambda_{\nu n} = \begin{cases} \dfrac{1}{N}\cos\left(\dfrac{n\nu\pi}{N}\right), n = 0 \\ \dfrac{2}{N}\cos\left(\dfrac{n\nu\pi}{N}\right), n = 1, 2, \cdots, N-1 \\ \dfrac{1}{N}\cos\left(\dfrac{n\nu\pi}{N}\right), n = N \end{cases}}$$

В матрични означения това е $\mathbf{a} = \mathbf{\Lambda f}$ и вземайки предвид (17) получаваме

(2.20) $$I_N^{CC} = \mathbf{W}^T \mathbf{\Lambda f}$$

където **f** е векторът с компоненти $f(t_n)$, n = 0, 1, ... , N, а $\mathbf{\Lambda}$ е матрицата с елементи описани в (2.20). В крайна сметка

(2.21) $$\boxed{I_N^{CC} = \mathbf{w} \cdot \mathbf{f}}.$$

Тук

(2.22) $$\boxed{\mathbf{w} = \mathbf{\Lambda}^T \mathbf{W}}$$

са теглата в квадратурната формула на Clenshaw-Curtis.

2.4. Използване на многоядрената архитектура.

2.4.1. Реализация за tanh-sinh схемата.

В схемата tanh-sinh възлите и теглата не зависят от подинтегралната функция и реализацията на паралелното им пресмятане е почти праволинейна. В подобен тип разпаралеляване обаче е тънкост използването на независими обекти и стриктното следене за манипулацията на текущата точност от съответните методи. То не трябва да борави с промени в подразбиращата се точност (глобална променлива за MPIR библиотеката). Ако на места е необходимо пресмятане с различна точност, това трябва да става локално при инициализацията на съответните променливи (mpf_init2(precision)). Основният фрагмент от инициализацията на абсцисите и теглата изглежда така:

```
cnt = Environment.ProcessorCount;
if (cnt >= 8) cnt = 8;
else if (cnt >= 4) cnt = 4;
else if (cnt >= 2) cnt = 2;
else cnt = 1;
.................
THSH thsh = new THSH(digits, maxlevel);

uint max = 20;
for (int k = 1; k < maxlevel; k++) max *= 2;
int chunk = (int)max / cnt;
var tasks = new List<Task<List<mpir.mpf_t>>>();

for (int idx = 0; idx < cnt; idx++)
{
    int ii = idx;
    uint idx1 = (uint)(ii * chunk);
    uint idx2 = (uint)((ii+1) * chunk);
    tasks.Add(
        Task<List<mpir.mpf_t>>.Run(
            () =>
            {
                return (new XW(thsh.mmax, thsh.nqeps, idx1, idx2)).xw();
            }
        )
    );
}
List<mpir.mpf_t> rall = new List<mpir.mpf_t>();
var whenAllTask = Task.WhenAll<List<mpir.mpf_t>>(tasks);
var final = whenAllTask.ContinueWith
(
    t =>
    {
        for (int i = 0; i < tasks.Count; i++)
        {
            rall.AddRange(t.Result[i]);
        }
    }, TaskContinuationOptions.OnlyOnRanToCompletion
);
final.Wait();
uint npmax = 0;
for (int k = 0; k < rall.Count / 2; k++)
```

```
    {
        thsh.xc.Add(rall[2 * k]);
        thsh.wc.Add(rall[2 * k + 1]);
        if (mpir.mpf_cmp(rall[2 * k + 1], thsh.nqeps) < 0) break;
        npmax++;
    }
```

Тук thsh се формира предварително и включва съответните списъци (празни) с абсциси и тегла (thsh.xc и thsh.wc). Накрая се определя и ефективният брой на необходимите за по-нататък абсциси и тегла npmax. XW е клас, чийто основен метод xw пресмята абсцисите и теглата в зададен интервал [idx1,idx2). Реализиран е доста икономично. Абсцисите и теглата са $x_k = \tanh\left(\frac{1}{2}\pi \sinh(kh)\right)$ и $w_k = \left(\frac{1}{2}h\pi \cosh(kh)\right) / \cosh^2\left(\frac{1}{2}\pi \sinh(kh)\right)$, но в цикъла се извикват само по един път sinh и exp (с различни аргументи). GetPi() се извиква многократно, но е реализирана така, че използва статично поле в класа MPMath, така че реално се пресмята стойност само ако се изисква по-голяма точност от тази, фиксирана в статичното поле.

```
    internal List<mpir.mpf_t> xw()
    {
        List<mpir.mpf_t> res = new List<mpir.mpf_t>();
        mpir.mpf_set_ui(h, 1);

        for (int i = 1; i <= m; i++)
        {
            mpir.mpf_div_ui(h, h, 2);
        }
        mpir.mpf_set_ui(test, 1);
        mpir.mpf_set(pi2, MPMath.GetPi());
        mpir.mpf_div_ui(pi2, pi2, 2);
        for (uint k = idx1; k < idx2; k++)
        {
            mpir.mpf_mul_ui(t, h, k); // t = k*h
            mpir.mpf_set(tx, MPMath.Sinh(t)); mpir.mpf_set(tn, tx);
            mpir.mpf_mul(tx, tx, pi2); // tx = (pi/2)*sinh(t)
            mpir.mpf_set(t2, MPMath.Exp(tx));
            mpir.mpf_ui_div(t1, 1, t2);
            mpir.mpf_add(t1, t2, t1);
            mpir.mpf_div_ui(t1, t1, 2); mpir.mpf_pow_ui(td, t1, 2);

            xel = mpir.mpf_init_set_ui(0);
            mpir.mpf_mul(t1, t1, t2);
            mpir.mpf_ui_div(xel, 1, t1);
            mpir.mpf_pow_ui(tn, tn, 2); mpir.mpf_add_ui(tn, tn, 1); mpir.mpf_sqrt(tn,
tn);
            mpir.mpf_mul(tn, tn, pi2); // (pi/2)*cosh(t)
            wel = mpir.mpf_init_set_ui(0);
            mpir.mpf_div(wel, tn, td);
            if (mpir.mpf_cmp(wel, eps0) < 0) break;
            else
            {
                res.Add(xel);
                res.Add(wel);
```

```
        }
    }
    return res;
}
```

Ако се правят пресмятания с една и съща точност за много задачи, резултатите от тази част могат да се запишат във файл.

След това се пресмятат стойностите на функцията в необходимите npmax точки. Тук отново се използва паралелно пресмятане. Използваме метода f_full() на обект от клас MFF, чието основно предназначение е именно да използва този метод с входни данни от вече формирания списък thsh.xc с абсциси, чиято дължина е npmax.

```
MFF mff = new MFF(npmax, maxlevel, thsh.xc);
List<mpir.mpf_t> f = mff.f_full();
.....................
internal List<mpir.mpf_t> f_full()
{
    int Nmax = 20;
    for (int k = 1; k <= maxlevel; k++) Nmax *= 2;
    int chunk = (int)nt / cnt;
    int chunk0 = (int)nt - (cnt - 1) * ((int)nt / cnt);
    int tchunk;

    Console.WriteLine("  cnt = {0}", cnt);
    Console.WriteLine("   nt = {0}",  nt);
    Console.WriteLine("chunk = {0}", chunk);

    List<mpir.mpf_t>[] fl = new List<mpir.mpf_t>[cnt];
    for (int n = 0; n < cnt; n++)
    {
        fl[n] = new List<mpir.mpf_t>();
        if (n < cnt - 1) tchunk = chunk;
        else tchunk = chunk0;
        for (int k = 0; k < tchunk; k++)
        {
            mpir.mpf_t t = mpir.mpf_init_set(x[n * chunk + k]);
            mpir.mpf_ui_sub(t, 1, t);
            //mpir.mpf_sub_ui(t, t, 1);
            fl[n].Add(t);
        }
    }
    for (int n = 0; n < cnt; n++)
    {
        if (n < cnt - 1) tchunk = chunk;
        else tchunk = chunk0;
        for (int k = 0; k < tchunk; k++)
        {
            mpir.mpf_t t = mpir.mpf_init_set(x[n * chunk + k]);
            mpir.mpf_sub_ui(t, t, 1);
            fl[n].Add(t);
        }
    }

    var tasks = new List<Task<List<mpir.mpf_t>>>();

    for (int idx = 0; idx < cnt; idx++)
```

```
        {
            int ii = idx;
            int ch = 2*chunk;
            if (ii == cnt - 1) ch = 2*chunk0;

            tasks.Add(
                Task<List<mpir.mpf_t>>.Run(
                    () =>
                    {
                        return (new FPS()).processing(ch, new
List<mpir.mpf_t>(fl[ii]),
                                    new Integrand(), ii);
                    }
                )
            );
        }

        List<mpir.mpf_t> rall = new List<mpir.mpf_t>();

        var whenAllTask = Task.WhenAll<List<mpir.mpf_t>>(tasks);

        var final = whenAllTask.ContinueWith(t =>
        {
                for (int i = 0; i < tasks.Count; i++)
                {
                    int chk = chunk;
                    if (i == cnt - 1) chk = chunk0;
                    for (int k = 0; k < chk; k++) rall.Add(t.Result[i][k]);
                }
                for (int i = 0; i < tasks.Count; i++)
                {
                    int chk = chunk;
                    if (i == cnt - 1) chk = chunk0;
                    for (int k = chk; k < t.Result[i].Count; k++)
                          rall.Add(t.Result[i][k]);
                }
        }, TaskContinuationOptions.OnlyOnRanToCompletion
        );
        final.Wait();

        return rall;
    }
```

Отново се използва отделен обект за всяка задача - FPS, в случая изключително прост, чийто метод processing използва независим за всяка задача обект от клас Integrand, съдържащ метода, пресмятащ подинтегралната функция.

```
    using fncs;

    ..............

    namespace fncs
    {
        delegate mpir.mpf_t func(mpir.mpf_t x);
        delegate List<mpir.mpf_t> func1(int len, List<mpir.mpf_t> l, Integrand I, int
num);
        delegate List<mpir.mpf_t> func2(int len, List<mpir.mpf_t> l, XW xw, int num);
    }
```

```
.............
internal class FPS
{
    internal func1 processing;

    internal FPS()
    {
        processing = FPS_processing;
    }

    internal List<mpir.mpf_t> FPS_processing(int len, List<mpir.mpf_t> l,
                                             Integrand I, int num)
    {
        List<mpir.mpf_t> res = new List<mpir.mpf_t>();
        for (int i = 0; i < len; i++)
        {
            res.Add(I.FNC(l[i]));
        }
        return res;
    }
}
```

Integrand за случая на тестов пример 1 от [18] изглежда така (вж. подраздела "Сравнителни тестове" по-надолу).

```
internal class Integrand // Ex.1 Int_0_1 t*ln(1+t) = 1/4
{
    internal mpir.mpf_t a, b;
    mpir.mpf_t x, tmp;

    internal Integrand()
    {
        a = mpir.mpf_init_set_ui(0);
        b = mpir.mpf_init_set_ui(1);

        x = mpir.mpf_init_set_ui(0);
       tmp = mpir.mpf_init_set_ui(0);
    }
    internal mpir.mpf_t FNC(mpir.mpf_t x0)
    {
        mpir.mpf_set(x, x0);
        mpir.mpf_add_ui(x, x, 1);
        mpir.mpf_div_ui(x, x, 2);
        mpir.mpf_t res = mpir.mpf_init_set_ui(0);

        mpir.mpf_add_ui(tmp, x, 1);
        mpir.mpf_set(res, MPMath.logSK(tmp));
        mpir.mpf_mul(res, res, x);

        //mpir.mpf_div_ui(res, res, 2);
        return res;
    }
}
```

След цялата тази предварителна подготовка се използва описаният вече алгоритъм tanh-sinh с една основна модификация, състояща се в това, че всички изчислявани там

стойности се вземат от вече готови списъци, а не се изчисляват на място. Тук не се използват паралелни пресмятания. Тези последни изчисления се свеждат единствено до умножение и събиране и са много малка част от времето, изисквано за цялото пресмятане.

2.4.2. Реализация за Clenshaw-Curtis схемата.

На първо място са ни необходими точките в интервала [-1,1] с координати $\cos\left(\frac{\nu\pi}{2^n}\right)$, където n е съответното ниво, а $\nu = 0,1,\ldots,2^n$. Една изключително бърза реализация е рекурсивното генериране (вж. [161], стр. 417), използващо само две извиквания на sin функцията. Ако $\varphi = 0$ и $\gamma = \frac{\pi}{2^n}$, то схемата изглежда така

$$c_0 = 1 /*\cos\varphi*/$$
$$s_0 = 0 /*\sin\varphi*/$$
$$\alpha = 2\left(\sin\left(\frac{\gamma}{2}\right)\right)^2$$
$$\beta = \sin(\gamma)$$
$$c_{k+1} = c_k - (\alpha c_k + \beta s_k)$$
$$s_{k+1} = s_k - (\alpha s_k - \beta c_k)$$

За предотвратяване загубата на точност може да се използва съвсем малко по-висока работна точност на пресмятанията. Ако digits отговаря на броя десетични знаци, digits + log(digits) е достатъчно. Тук не се използват паралелни пресмятания. Рекурсивното генериране е изключително бързо и заема много малка част от времето на цялостното пресмятане.

За пресмятането на теглата се използва схема с N*log(N) операции (тук $N = 2^n$, n е нивото), публикувана в [162]. Същината се състои в генерирането на вектор (използващ единствено получените вече по-горе абсциси), за който се използва бързо обратно Фурие преобразование, даващо теглата. Тук също не се използват паралелни пресмятания.

Паралелното пресмятане на стойностите на подинтегралната функция изглеждат по същия начин, като описания за tanh-sinh схемата. И отново за пресмятанията даващи последователните нива не е необходимо паралелно пресмятане.

2.4.3. Сравнителни тестове.

Тестовете са извършени на преносим компютър със следните характеристики: процесор Intel (R) Core (TM) i7-3610QM CPU @ 2.30GHz (4 физически процесора, 8 логически процесора (нишки) чрез hyper-thraeding технология; до 3.1 GHz при 4 активни процесора чрез Turbo Boost). 16 GB RAM, 64-битова операционна система Windows 7 Enterprise (Microsoft Windows NT 6.1.7601 Service Pack 1). В таблиците по-долу е отбелязан като TL (test laptop).

За база на изследването се взема [18]. Там са приведени 14 примерни интеграла, изчислявани с точност 2000 десетични знака. Сравняват се получените резултати от

разработените от нас програмни инструменти THSHPar (схема с паралелни пресмятания за tanh-sinh квадратура) и CCPar (схема с паралелни пресмятания за Clenshaw-Curtis квадратура) с някои от резултатите получени в [18]. Тъй като програмата CCPar използва квадратурна схема на Clenshaw-Curtis, данните за нея са само там, където тя е ефективно приложима: необходимо условие е подинтегралната функция и производните ѝ да са непрекъснати, с крайни стойности за целия интервал.

Примерните интегрални са следните

1. $\int_0^1 t\log\left(1+t\right)\mathrm{d}t = 1/4$

2. $\int_0^1 t^2\arctan(t)\,\mathrm{d}t = \left(\pi - 2 + 2\log(2)\right)/12$

3. $\int_0^{\pi/2} e^t\cos(t)\,\mathrm{d}t = \left(e^{\pi/2}-1\right)/2$

4. $\int_0^1 \frac{\arctan\left(\sqrt{2+t^2}\right)}{(1+t^2)\sqrt{2+t^2}}\mathrm{d}t = 5\pi^2/96$

5. $\int_0^1 \sqrt{t}\log(t)\,\mathrm{d}t = -4/9$

6. $\int_0^1 \sqrt{1-t^2}\,\mathrm{d}t = \pi/4$

7. $\int_0^1 \sqrt{\frac{t}{1-t^2}}\,\mathrm{d}t = 2\sqrt{\pi}\Gamma(3/4)/\Gamma(1/4)$

8. $\int_0^1 \log^2(t)\,\mathrm{d}t = 2$

9. $\int_0^{\pi/2} \log\left(\cos(t)\right)\mathrm{d}t = -\pi\log(2)/2$

10. $\int_0^{\pi/2} \sqrt{\tan(t)}\,\mathrm{d}t = \pi\sqrt{2}/2$

11. $\int_0^{\infty} \frac{1}{1+t^2}\mathrm{d}t = \pi/2$

12. $\int_0^{\infty} \frac{e^{-t}}{\sqrt{t}}\mathrm{d}t = \sqrt{\pi}$

13. $\int_0^{\infty} e^{-t^2/2}\,\mathrm{d}t = \sqrt{\pi/2}$

14. $\int_0^{\infty} e^{-t}\cos(t)\,\mathrm{d}t = 1/2$.

		Брой процесори			
Пример	Изисквано ниво	4, [18]	16, [18]	1024, [18]	4, THSHPar+TL
Инициализация за ниво 13		1085.34	271.87	5.02	382.31
1	10	101.63	25.55	0.53	6.11
2	10	294.32	74.04	1.54	129.03
3	10	317.01	79.69	1.83	44.75
4	10	328.73	82.13	1.63	130.31
5	9	51.62	12.90	0.30	4.69
6	10	5.62	1.42	0.05	0.43
7	10	11.46	2.87	0.10	0.65
8	9	50.98	12.85	0.27	4.70
9	10	333.24	83.60	1.84	17.54
10	10	245.45	61.39	1.44	11.84
11	11	5.17	1.30	0.04	1.67
12	12	161.99	40.71	0.80	112.47
13	13	216.50	54.13	0.97	214.40
14	13	1826.02	457.02	7.87	309.41
общо, без инициализацията		3949.74	989.6		988.00
корекция за различния хардуер в [18]	1/[(3.1/2)* 1.3]	1960.17	492.12		988.00

Таблица 2.2. Сравнителни резултати за THSHPar и набор от примери в [18].

В третия ред на таблица е приведено времето за инициализация, което не се включва по нататък. В следващите редове е дадено времето за изпълнение без инициализацията. В последния ред е добавена корекция, отчитаща максималното възможно ускорение спрямо използвания в [18] хардуер, имайки предвид максимално достиганата тактова честота с Turbo Boost и максималния процент подобрение с около 30%, даван от hyper-threading технологията в TL. Тук, естествено, са отчетени само очевидните фактори. Би могла да оказва влияние и бързината на използваната основна RAM памет, а и тази за кеша на процесорите, за които не разполагаме за съжаление с данни за сравнение спрямо [18]. При такива данни (без включване на времето за инициализация и в двете реализации, тази в [18] и нашата), груба сметка (492,12*2 = 982,24 ~ 988.00) показва, че резултатите,

постигнатите от нас с разпаралеляване на 4 процесора от TL, са еквивалентни на 8 процесора, от основната схема в [18]. Без корекцията, резултатът е равен на 16 процесора от [18].

За пример 14 при 13 нива не се достигат желаните 2000 знака, а около 1971 (и в [18], и за THSHPar+TL). В [18] се казва, че тези 2000 знака са достижими с 14 нива, без да се посочват данни. Това наистина е така. THSHPar+TL постигат тази точност с 14 нива за 620.90 секунди пресмятания плюс 768.55 секунди инициализация за 14 нива.

			Брой процесори					
Пример	Изисквано ниво в [18]	Изисквано ниво в CCPar	4, [18]	16, [18]	64, [18]	256, [18]	1024, [18]	4, CCPar +TL
(Инициали-зация за ниво 13)/8, т.е. за ниво 10			135.67	33.98	8.61	2.22	0.63	0 надолу се дава цялото време
1	10	12	101.63	25.55	6.45	1.65	0.53	1.84
2	10	12	294.32	74.04	18.83	4.99	1.54	10.95
3	10	10	317.01	79.69	20.42	5.24	1.83	1.23
4	10	12	328.73	82.13	20.84	5.52	1.63	10.99
Общо без иниц.			1041.69	261.41	66.54	17.40	5.53	25.01
Общо с иниц.(всеки път), т.е + (ред 3)*4			1584.37	397.33	100.98	26.28	8.05	25.01
Общо с корекция за хардуера			786.29	197.19	50.11	13.04	4.00	25.01

Таблица 2.3. Сравнителни резултати за CCPar и част от набора от примери в [18].

В предпоследния ред на таблица 2.3 е приведено времето, изисквано за изпълнение и на четирите примера, при условие, че инициализацията се извършва всеки път. Там, където е приложима, паралелната Clenshaw-Curtis квадратура (с 4 процесора: CCPar + TL) е еквивалентна на около 128 процесора от системата използвана в [18]. Това е с корекция за хардуера, както по-горе. Иначе се равнява на около 256 процесора. Тук обаче и приложена друга схема и такова сравнение е донякъде спекулативно.

Има разлика за означаването на нивата в двата вида схеми, която трябва да бъде отбелязана. В tanh-sinh схемата ниво k означава $20*2^k$, а при Clenshaw_Curtis схемата ниво k означава 2^k. Тук използваният метод за пресмятане на теглата, включващ бързо обратно Фурие преобразование, изисква броят точки да е степен на двойката.

Както бе казано вече, времето за решаване на конкретна задача, зависи от много фактори. При сравняване, някои от тях са неизвестни. Това не касае само хардуера. Много важен е дори и начинът на реализация на самите елементарни функции, участващи в израза за подинтегралната функция. Като пример може да се приведе еволюцията на

самата програмна система, представяна тук. В раздела за програмния инструмент NQTS бе споменато, че приведения пример за интеграл $\int_0^{\pi/2} \frac{\arcsin\left(\sqrt{2}/2 \cdot \sin x\right) \sin x}{\sqrt{4 - 2\sin^2 x}} \mathrm{d}x$, чиято аналитична оценка затруднява разпространените специализирани среди за математически пресмятания, след "продължителни пресмятания" дава резултата с точност 1000 десетични знака. Тези "продължителни пресмятания", в онзи момент (около две и половина години преди разработването на THSHPar и CCPar) всъщност отнемаха около 4 часа и половина на същия преносим компютър (TL). Към този момент THSHPar решава тази задача за 49.51 секунди (изискват се 11 нива, т.е. $N = 20 * 2^{11} = 40960$). От тях инициализация на абсцисите и теглата: 9.53 секунди. Пресмятане на подинтегралната функция в необходимите npmax = 8177 точки: 39.90 секунди. Разликата е много голяма и изисква обяснение. Още по-голяма е разликата с постигания от CCPar резултат за същата задача: общо 3.87 секунди (изискват се 11 нива, т.е. $N = 2^{11} = 2048$ точки). Голямата разлика се дължи на подобренията на всички фактори, участващи като множители в резултата време на постигания резултат. За THSHPar: Не се използва интерпретатор. В схемата се използва повишена точност, тогава, когато е нужно и броят на необходимите точки за пресмятане на подинтегралната функция е намален - условието за прекъсване в основната схема на алгоритъма, формиращ параметъра npmax. Използват се паралелни пресмятания. Има подобрения в реализацията на atan функцията (asin се пресмята чрез нея) и тригонометричните функции. Самата xmpir библиотека реализира по-нова версия на MPIR. Освен това, което е още по-важно, е оптимизирана за конкретния процесор. Всичко това (с изключение на подобряването на схемата за пресмятане на npmax) е валидно и за CCPar. Освен това, тази схема е по-подходяща за конкретния пример. Това не омаловажава качествата на tanh-sinh схемата. Тя е приложима там, където Clenshaw-Curtis схемата не е. Там, където има особености в края на интервала, than-sinh преобразованието ги "регуляризира, пращайки ги в безкрайност". Нещо, което Clenshaw-Curtis схемата в класическия си вид не може. Като любопитен факт обаче, тя е приложима и за примерите 13 и 14, да речем, макар и резултатите да не са впечатляващи. Използвайки преобразованието $\int_{-1}^{1} f\left(\frac{2x}{1-x} + 1\right) \frac{2}{(1-x)^2} \mathrm{d}x = \int_0^{\infty} f(x) \mathrm{d}x$, CCPar решава тези примери съответно за 60.62 и 652.43 секунди (изискват се съответно 16 и 19 нива). От формална гледна точка проблемът, разбира се, е в това, че интервалът за примерите е безкраен и макар че подинтегралната функция и производните се държат добре в него, оценката за грешката (включваща като множител дължината на интервала) става неопределена. Преобразованието към краен интервал, естествено не решава проблема, а го отмества. За пример 14 това е почти очевидно. При преобразованието производните на подинтегралната функция вече не са с ограничена вариация. Интеграли от такъв тип изискват специален подход за ефективното си решаване.

2.5. Заключение.

Изследвани са две от най-перспективните квадратурни схеми, предлагащи възможност за ефективно пресмятане на определени интеграли с висока точност. Предложена е методика на реализация, отчитаща особеностите на използваната среда на разработка, включително за паралелни пресмятания. Разработени са програмни инструменти за пресмятане на определени интеграли с висока точност. Един от тях (NQTS) демонстрира възможностите

за създаване на интерактивно графично приложение в използваната среда. За другите два (THSHPar и CCPar) са предложени схеми на реализация, позволяващи паралелни пресмятания. За тази реализация са направени тестове, показващи възможността за ефективни пресмятания с много висока точност със специфичните средства на използваната среда, включваща масово вградените в съвременните преносими и настолни компютри многоядрени процесори.

Глава 3. Методи за подобряване на численото решаване на ОДУ с висока точност.

3.1. Вместо увод: Относно смисъла на пресмятането с много голяма точност за примера на детерминистични модели с хаотично поведение.

Ще се разглежда случая на детерминистична дисипативна динамична система, притежаваща хаотичен атрактор. Разглежда се системата в точно определен смисъл, а именно динамична система. Без да се влиза в технически подробности, това е система, чието бъдеще (поне) е определено от настоящия момент. На математически език това е действие на полугрупа върху множество – в най-интересните случаи това множество има някаква структура, поне топологично пространство. „Действащата" полугрупа е „времето" - може да е дискретна или непрекъсната. Самото множество също може да е дискретно, пример са клетъчните автомати. Тук се разглежда частен, но много важен случай на система автономни обикновени диференциални уравнения. В този случай времето е непрекъснато, множеството е област в $\mathbb{R}^n$, а самото действие е потокът, т.е. решението на системата за различни начални условия, и се състои в преместване на точките от началното състояние при $t = 0$ до решението в момент $t = \tau$. Ако решенията са определени навсякъде, времето е група (в случая е $\mathbb{R}$) и системата е обратима. Бъдещето и миналото са определени еднозначно.

Въпросът има разбира се различни аспекти. Първи: До каква степен моделът е близък до реалността, за да се налага да бъде изчисляван с висока точност, поне за известен интервал (има се предвид динамична система)? Втори: Адекватни ли са математическите средства, използвани за моделирането и най-вече за решаването?

Двата въпроса са свързани до известна степен. Оставен е настрана философския въпрос какво всъщност се изучава – някаква реалност или съзнанието, създаващо модели на реалността, което не е априори едно и също. На първия въпрос може да се отговори в зависимост от това на каква позиция заставаме. От математическа гледна точка няма проблеми при формулирането. От физическа гледна точка, гладките динамични системи (с или без хаотично поведение) не са съвсем точни. Моделът престава да бъде точен за размерности съизмерими с тези, които са типични за дискретната природа на материята. Това от формална гледна точка. От друга страна, не е нужно моделът да е съвсем точен, за да отразява качествени страни на поведението на системата, което се случва много често. По-интересен е вторият въпрос. Тук неминуемо трябва да направим отклонение, за да формулира способностите си за решаване на подобни модели. Какво е характерно за тях? Без да има консенсус по точното определение на хаоса в с-ми с атрактор (хаотичен; има примери за странни, но не хаотични атрактори; изключваме консервативни системи, при които също е възможно хаотично поведение, но без атрактор), все пак има две неща които се смятат за съществени: 1) Чувствителност към началните условия и 2) Наличие (в атрактора) на топологически транзитивна орбита (т.е. съществува орбита, която минава през произволна околност на произволна точка на атрактора на системата). Грубо казано, първото условие е необходимо (без да е достатъчно), а второто се приближава до това да дава достатъчност. Някои автори добавят изискване за наличие на гъсто множество от периодични траектории, но това би могло да се окаже и излишно – изследвайки маса модели от позиции на символната динамика, това е така, но не ясно дали този признак не е вторичен.

И така, какво следва от наличието на топологически транзитивна орбита в контекста на реално пресмятане на модела? Отговорът е еднозначен: липса на възможност за точно пресмятане на орбита, имайки предвид крайната точност на пресмятанията. Това е очевидно. Топологически транзитивната орбита минава в произволна близост до всяка точка на атрактора. Следователно, колкото и да е голяма (но крайна) точността на пресмятанията, неминуемо идва моментът, в който пресмятаната орбита и тази топологически транзитивна орбита са неразличими за пресмятането (известната shadowing lemma може да се разглежда като друго представяне на този факт). Е, адекватни ли са средствата за пресмятане? Всъщност и да, и не. Разделителната линия е чисто математическа. Дори ако се задоволим с пресмятания само с изчислими (конструктивни) реални числа, тяхното поле е "конструктивно" пълно. От практическа гледна точка на реалните (физически – с изчислителна система) пресмятания обаче, нещата стоят по друг начин. Изчислителните системи могат потенциално да дадат произволна (но все пак крайна) точност на всяко едно конструктивно (изчислимо) реално число. За следване на точната орбита обаче, това не е достатъчно, поради описаната по-горе причина. Всъщност изчислителната система борави само с крайно подмножество на рационалните числа. Често някои автори опитват да свържат понятието "хаос" на детерминистична система с невъзможността за точно предсказване. От казаното по-горе става ясно, че това не е точно така от математическа гледна точка. Въпросът е в разминаване на средствата с които формулираме модела (идеален) и тези, с които разполагаме за приближено пресмятане (реално). Да, всичките ни пресмятания са с краен брой символи. Ако се поставя така въпросът, то ние не сме способни да пресмятаме орбитите (идеалните) по простата причина, че не можем да задаваме точно дори началната им точка (примерно една от координатите е ирационално число). Освен това предсказуемостта е нещо относително. Никога не можем да бъдем сигурни например, че даден сложен (природен, примерно) процес не е периодичен, тъй като не можем да го проследим до безкрайност. Конкретен пример за възможностите за пресмятане с все по-голяма точност е приведен в глава 7.

3.2. Обзор на неявните схеми на Рунге-Кута.

Тук са представени някои основни факти, необходими за построяването на процедура от тип неявна схема на Рунге-Кута.

Предполагайки, че $y(t)$ е реален m-мерен вектор, нека е дадена системата $\dot{y} = f\left(t, y(t)\right)$, за която искаме да решим задачата с начални стойности (задача на Коши) за $0 \le t \le T$ и $y(0) = y_0$.

3.2.1. Структура на методите Рунге-Кута.

Едностъпковите методи Рунге-Кута се описват с неявна зависимост между y_n и y_{n+1}, които са приближените стойности за последователните стъпки:

(3.1) $$y_{n+1} = y_n + \tau\,\Phi(\tau, y_n, y_{n+1}) \quad \tau = t_{n+1} - t_n \quad (n \ge 1),$$

където с y_n е означено приближението на $y(t)$ за t_n. Стъпката τ може да се променя с n.

Даден едностъпков метод Рунге-Кута се описва с формулата

(3.2) $$y_{n+1} = y_n + \tau \sum_{i=1}^{s} b_i k_i ,$$

където

(3.3) $$k_i = f\left(t_n + c_i\tau, y_n + \tau \sum_{j=1}^{s} a_{ij} k_j \right).$$

Формулата (2) е s-етапна. В случая, когато $a_{ij} = 0$ за $j \geq i$, коефициентите k_i могат явно да се пресметнат от стойностите на $k_1, \cdots, k_{i-1}$. Подобна формула се нарича *явна*. В случая $a_{ij} = 0$ за $j > i$, но $a_{ii} \neq 0$, всяко k_i е неявно определено от уравнението

(3.4) $$k_i = f\left(t_n + c_i\tau, y_n + \tau \sum_{j=1}^{i-1} a_{ij} k_j + \tau a_{ii} k_i \right).$$

Такива методи Рунге-Кута се наричат *диагонално неявни* (*diagonally implicit)*. Изпълнението на една стъпка на подобен метод изисква решаването на система от *s* нелинейни алгебрични уравнения (всяка система е от ред *m*), което обикновено се прави с помощта на някаква итеративна процедура. В общия случай (не е явен и не е диагонално неявен) методът е *неявен*. Всички k_i трябва да се изчисляват едновременно. По такъв начин, една стъпка на неявен метод изисква решаването на нелинейна алгебрична система от ред ms. Отново трябва да се ползва итеративна процедура.

Коефициентите във формулите (2) и (3) обикновено удовлетворяват условието

(3.5) $$c_i = \sum_{j=1}^{s} a_{ij} \quad (1 \leq i \leq s).$$

За представяне на даден метод Рунге-Кута (посредством коефициентите му) се използва така наречената таблица на Бътчер:

(3.6)
$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array} \quad \left(\text{или съкратено} \quad \frac{c \mid A}{\mid b} \right)$$

3.2.2. Порядък на апроксимацията. Опростяващи условия.

Приемайки, че

(3.7) $$\hat{l}_{n+1} = y(t_{n+1}) - \hat{y}_{n+1}, \quad \hat{y}_{n+1} = y(t_n) + \tau\, \Phi\left(\tau, y(t_n), \hat{y}_{n+1}\right)$$

бележи локалната грешка от дискретизация (грешка от прекъсване), порядъкът на апроксимацията се определя като най-голямото неотрицателно цяло число *p*, за което

(3.8) $$\hat{l}_{n+1} = O\left(\tau^{p+1}\right), \quad \tau \to 0.$$

За разглежданите методи порядъкът на апроксимацията може да се определи с

помощта на опростяващите условия на Бътчер, приведени по-долу (от *A* до *E*)

$A(\xi)$, ако $\hat{l}_{n+1} = O\left(\tau^{\xi+1}\right)$

$B(\xi)$, ако $\sum_{i=1}^{s} b_i c_i^{k-1} = \frac{1}{k} \quad (1 \le k \le \xi)$

$C(\xi)$, ако $\sum_{j=1}^{s} a_{ij} c_j^{k-1} = \frac{1}{k} c_i^k \quad (1 \le k \le \xi, 1 \le i \le s)$

$D(\eta)$, ако $\sum_{i=1}^{s} b_i c_i^{l-1} a_{ij} = \frac{1}{l} b_j \left(1 - c_j^l\right) \quad (1 \le j \le s, 1 \le l \le \eta)$

$E(\xi,\eta)$, ако $\sum_{i=1}^{s} \sum_{j=1}^{s} b_j c_i^{l-1} a_{ij} c_j^{k-1} = \frac{1}{k(l+k)} \quad (1 \le l \le \eta, 1 \le k \le \xi)$.

Фигура 3.1. Опростяващи условия на Бътчер.

Условието $A(\xi)$ означава, че методът има порядък на апроксимацията не по-малък от ξ. Прилагането на метод на Рунге-Кута към диференциалното уравнение $\dot{y}(t) = f(t)$, $y(t_n) = 0$ дава квадратурната формула

(3.9) $$y_n = \tau \sum_{i=1}^{s} b_i f(t_n + c_i \tau),$$

чиято дясна част е апроксимация на интеграла

(3.10) $$y(t_{n+1}) = \tau \int_0^1 f(t_{n+1} + x\tau)\,\mathrm{d}x .$$

Методът за числено интегриране на (8) се определя изцяло от абсцисите c_i и теглата b_i на метода Рунге-Кута. Следователно условието $B(\xi)$ означава, че квадратурната формула е точна ако f р полином от степен не по-висока от $\xi - 1$. Това е еквивалентно на твърдението, че квадратурната формула (8) в от порядък ξ.

Следните резултати (получени от Бътчер) отразяват зависимостите между условията.

Ако за даден метод Рунге-Кута всички абсциси $c_1, \ldots, c_s$ са различни и всички тегла $b_1, \ldots, b_s$ са различни от нула, то са изпълнени следните отношения:

$$A(\xi) \Rightarrow B(\xi),$$
$$A(\eta+\xi) \Rightarrow E(\eta,\xi),$$
$$B(\eta+\xi) \wedge C(\xi) \Rightarrow E(\eta,\xi),$$
$$B(\eta+\xi) \wedge D(\eta) \Rightarrow E(\eta,\xi),$$
$$B(s+\xi) \wedge E(s,\xi) \Rightarrow C(\xi),$$
$$B(\eta+s) \wedge E(\eta,s) \Rightarrow D(\eta),$$
$$B(p) \wedge C(\xi) \wedge D(\eta) \Rightarrow A(p) \quad (p \le \min(\xi+\eta+1, 2\xi+2)).$$

При същите условия са валидни и следните отношения:

$$A(2s) \Rightarrow B(2s) \wedge C(s) \wedge D(s),$$
$$B(2s) \wedge C(s) \Rightarrow D(s),$$
$$B(2s) \wedge D(s) \Rightarrow C(s),$$
$$B(2s) \wedge C(s) \wedge D(s) \Rightarrow A(2s).$$

А ето и резултат [49], който дава възможност за пресмятане на коефициентите a_{ij} в таблица на Бътчер:

Нека са дадени s различни абсциси $c_1, \ldots, c_s$. Тогава условията $B(s)$ и $C(s)$ определят еднозначно схема на Рунге-Кута. Това е вярно и за условията $B(s)$ и $D(s)$. Методът, който се определя от условията $B(s)$ и $C(s)$, има порядък на апроксимацията не по-малък от s.

Порядъкът на апроксимацията може да е по-голям от s при подходящ избор на абсцисите. По-специално, това е така, когато абсцисите са избрани да бъдат възли на квадратурна формула от висок порядък. Ще обсъдим случая на s-етапен метод на Гаус-Лежандър, чиито абсциси са корени на модифицираните полиноми на Лежандър $P_s^*(x)$, определени в интервала [0,1] ([163]). Ако определим матрицата V като

$$V = \begin{bmatrix} 1 & c_1 & \cdots & c_1^{s-1} \\ 1 & c_2 & \cdots & c_2^{s-1} \\ \vdots & \vdots & & \vdots \\ 1 & c_s & \cdots & c_s^{s-1} \end{bmatrix},$$

и диагоналната матрица S като

$$S = \begin{bmatrix} 1 & & & \\ & \frac{1}{2} & & \\ & & \ddots & \\ & & & \frac{1}{s} \end{bmatrix},$$

то теглата за този метод се определят от условието $B(s)$ по следния начин:

$V^{\mathrm{T}} b = Se$,

където b е векторът на теглата, а $e = [1,1, \ldots, 1]^{\mathrm{T}}$.

Останалите параметри се определят от условието $C(s)$. Построеният по този начин метод удовлетворява условието $B(2s)$ и от втория от цитираните по-горе резултати следва, че s-етапният метод на Гаус-Лежандър има порядък на апроксимацията $2s$.

3.3. Реализация на програмни инструменти за решаване на системи обикновени диференциални уравнения.

Изградените програмни инструменти са конзолен, за общ случай на система от обикновени диференциални уравнения в нормална форма (явна зависимост на производните) и графичен за частен, но интересен случай на едно уравнение от втори ред.

3.3.1. Реализация на конзолен програмен инструмент за решаване на системи обикновени диференциални уравнения.

Конзолният програмен инструмент изисква отделен файл, описващ задачата. Всъщност единствената "автоматизация" е на ниво произволно име на инициализиращия файл при компилация със спомагателен пакетен файл, чието съдържание е нещо от вида

```
csc  /debug:full IRK_Test.cs %1.cs funcs.cs xmpir.cs.
```

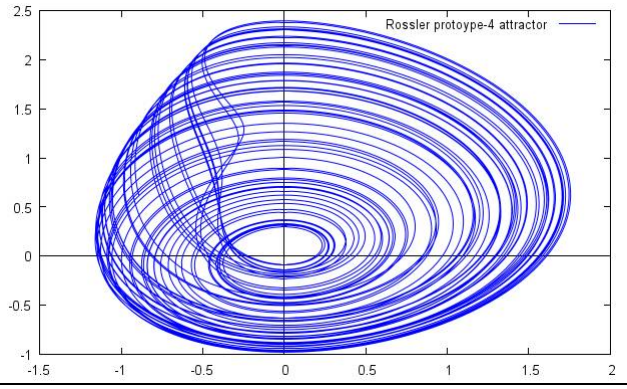
Първият аргумент от командния ред за получения по този начин изпълним файл е управляващ режима (решаване, визуализация) и следващите се интерпретират според него. В случай на решаване се задават допълнително ред на неявната схема, точност и брой стъпки. Резултатът е изходен файл на решението с фиксирано име irktest.dat. Началната точка е част от задаваната информация в допълнителния файл за инициализация (зададен за %1). За това има очевидна причина. Част от компонентите на точката може да са някакви приближения на математически константи, чието записване на ръка като вход дори само със 100 знака би било крайно неудобно. С това приложение, бяха осъществени редица тестове. Някои от тях са изведени в отделна глава 7. Тук ще покажем само три.

Система на Рьослер, прототип 4

$$\begin{cases} \dot{x} = -y - z \\ \dot{y} = x + ay \\ \dot{z} = b + z(x - c) \end{cases}$$

$$(a,b,c) = (0.5,1,3)$$

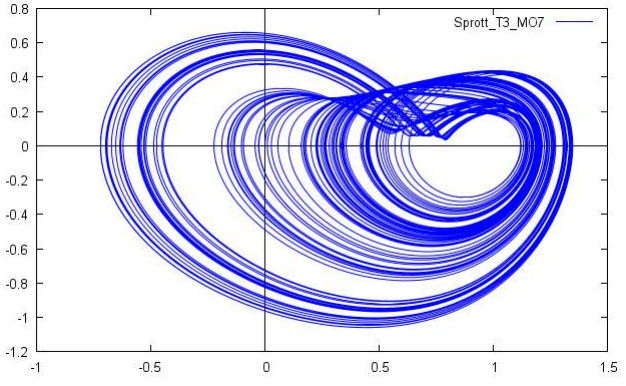
$$(x_0, y_0, z_0) = (2,0,1)$$



Осцилатор с памет

$$\dddot{x} + 0.6\ddot{x} + \dot{x} = x^2\left(1 - x^2\right)$$

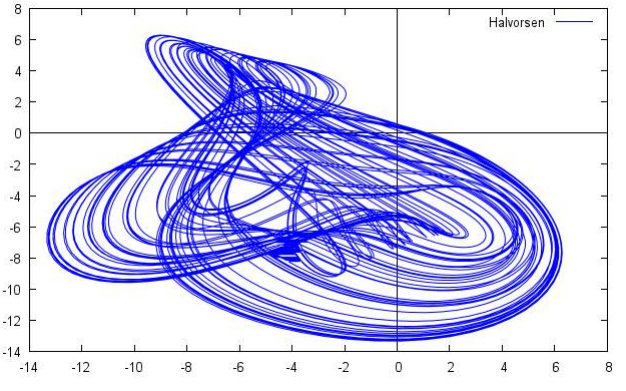
$$(\ddot{x}_0, \dot{x}_0, x_0) = (0,0,0.4)$$



Циклична система $(Halvorsen)$

$$\begin{cases} \dot{x} = 1.3x - 4y - 4z - y^2 \\ \textit{плюс циклична замяна} \\ (x,y,z) \to (y,z,x) \to (z,x,y) \end{cases}$$

$$(x_0, y_0, z_0) = (-6.4,0,0)$$



Фигура 3.2. Три примера за получени и визуализирани с IRK_Test решения.

Възможно е допълнително усъвършенстване на интерфейса. Съществува например възможност за използването му по специфичен начин, запазвайки ефективността (без използване на интерпретатор в прекия смисъл) и съставяйки от въведения модел C# файл, който се компилира заедно с останалите компоненти "зад кадър" до изпълним файл за конкретната задача, преди изпълнение. Това бе направено в малко по-различен контекст преди няколко години (оптимизационни задачи).

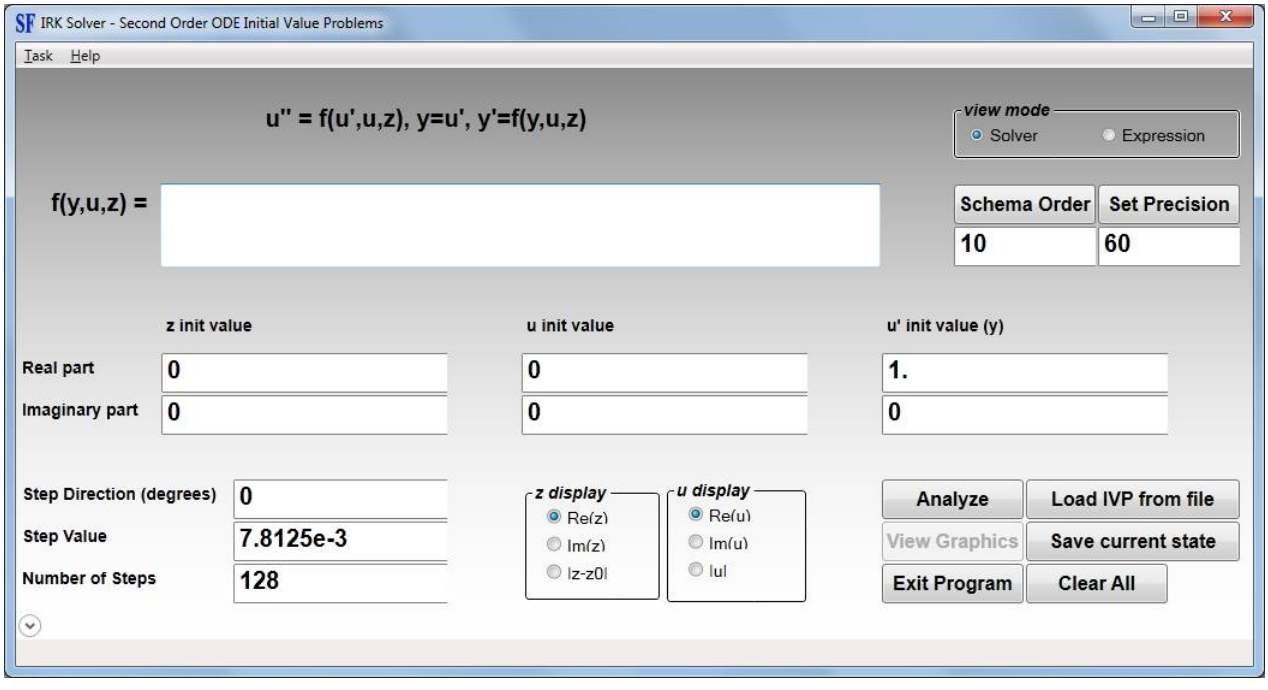
Освен основния вариант на решател (class IRK_Solver), системата разполага с още един (class ODEX), реализиращ екстраполационен метод. За база е взет програмният код та известна фортранска програма ([76],[66]), който е адаптиран за C# + XMPIR, като параметрите му са зададени за произволна точност. По-специално параметрите 'km' и 'uround', както и текущата точност на пресмятанията се определят в зависимост от изискваната точност на решението 'tol'. Освен това е променено поведението на параметъра 'iout' в случая iout=1, така че изходните точки се получават на интервал, не по-

голям от желана стойност (нов входен параметър). Това се налага поради факта, че тази стойност на iout е основна, когато изискваната точност е висока. В този случай използването с iout=2 (dense output) води до много малки стойности на стъпката h, поради вградената оценка на грешката от интерполация.

3.3.2. Реализация на специален графичен програмен инструмент VODE2 за решаване на ОДУ от втори ред.

Програмният инструмент е предназначен за числено изследване с визуализация на решението на обикновено диференциално уравнение от втори ред $\mathbf{u'' = f(u', u, z), (' = d/dz)}$. Той решава задача начални стойности $\mathbf{z = z_0}$, $\mathbf{u_0 = u(z_0)}$, $\mathbf{u'_0 = u'(z_0)}$, наречена тук IVP (Initial Value Problem). Уравненията могат да са в реална или комплексна област. Пресмятанията се извършват с произволна, задавана явно, точност. Генерираното решение се показва графично в процеса на решаването. Изразът за уравнението се дава в текстова форма и може да включва алгебрични операции и функции. Предоставена е възможност за записване на текущото състояние на задачата, както и за прочитане на записано преди това състояние на задача от файл. Програмата може да работи в така наречения режим на израз, т.е. да изчислява и визуализира израз (функционален). И в двата режима са допустими допълнителни параметри. Интерпретаторът приема произволен брой променливи, но тези извън формулировката на основната задача се считат за параметри, за чиято инициализация се изисква отделна инициализация от потребителя. Визуализацията се извършва с помощта на програмата gnuplot. Всички пресмятания се извършват с комплексни числа (произволна точност).

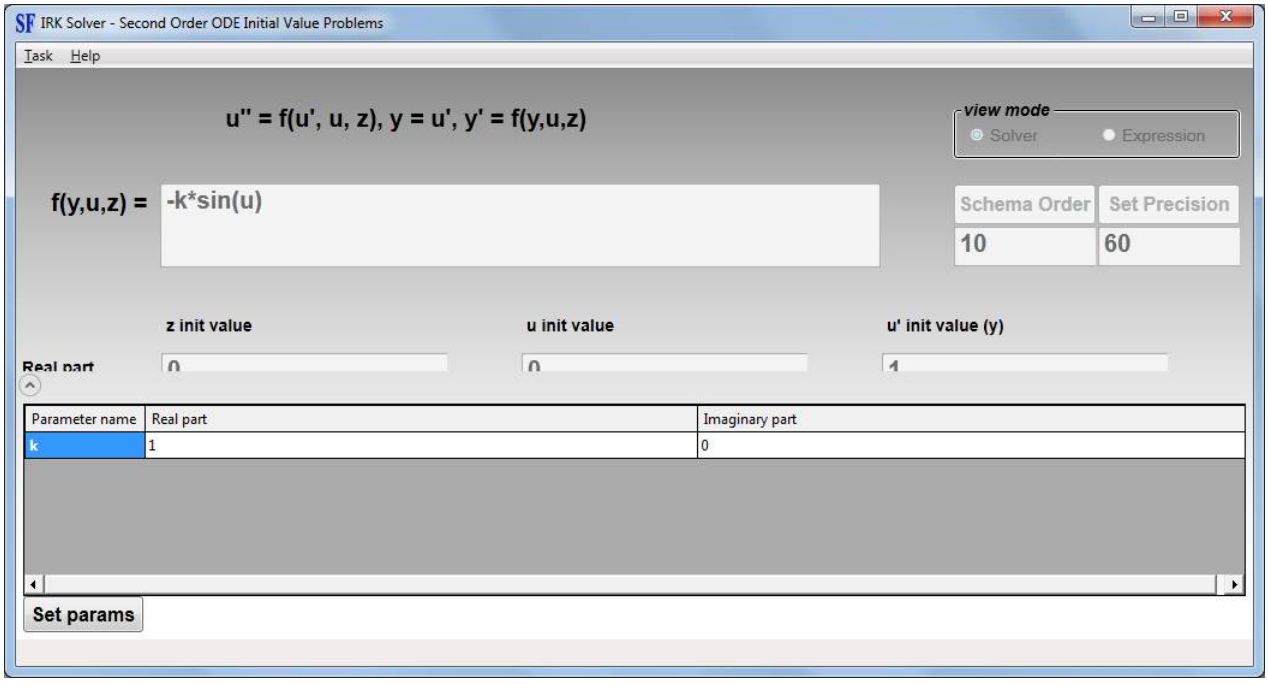
Конкретните действия могат да се инициират от менюто или чрез съответния бутон. В зависимост от етапа на формулирането/решаването на задачата, е достъпна различна част от интерфейса. В процеса на решаване е възможно достигане на особена точка. В този случай се показва съобщение "Apparently a pole reached.", което не е съвсем точно (точката може да е съществена особеност, а не полюс), но дава идея за възникналия проблем. В този случай процесът на решаване завършва. Визуализацията се извършва чрез пренасочване на стандартния изходен поток към предварително стартираната в отделен процес програма gnuplot. Имената на променливите, които са част от уравнението (или функционалния израз) са фиксирани, за да се избегнат усложнения при формулиране на задачата. Освен тези променливи, в израза могат да присъстват и други променливи (параметри). Когато са налични параметри, програмата изисква от потребителя да ги инициализира отделно. В основния израз са допустими само реални константи. Ако е необходима комплексна константа, тя може да се включи като параметър и да и се присвои стойност при инициализацията на този параметър. Основното ограничение на програмата е, разбира се, типът на решаваната задача (по принцип използваният решател е за система от n обикновени диференциални уравнения от първи ред в нормална форма). Все пак дори тази задача (ОДУ от втори ред) в общия си вид е достатъчно интересна. Самата идея за създаване на този програмен инструмент възникна от желанието за числено изследване на решения на някои уравнения на Пенлеве.



Фигура 3.3. Програмата VODE2 след първоначално стартиране.

Повечето от предвидените в интерфейса полета са ясни без коментар. Полетата за визуализация задават кои компоненти се визуализират. Преди стартиране на (графичното представяне) на решението, трябва да се анализира формалната валидност на въведения за решаване израз (Analyze). При липса на грешки, бутонът View Graphics става достъпен и решаването може да започне. В противен случай се дава съобщение за възникналата грешка. В случай на използване на програмата като визуализатор на функция (избора на Expression вместо Solver) полето Number of steps се преименува на Number of points. Основните променливи на задачата са z, u и y. Когато присъстват променливи с други имена, те се считат за параметри (вж. по-надолу). Променливата y играе ролята на u' в първоначалното уравнение.

Въвеждане на параметри. Ако въведеният израз има променливи, които са различни от основните променливи на задачата, след иницииране на проверка с “Analyze”, потребителят бива подканен да въведе стойности за тези променливи (параметри), например:

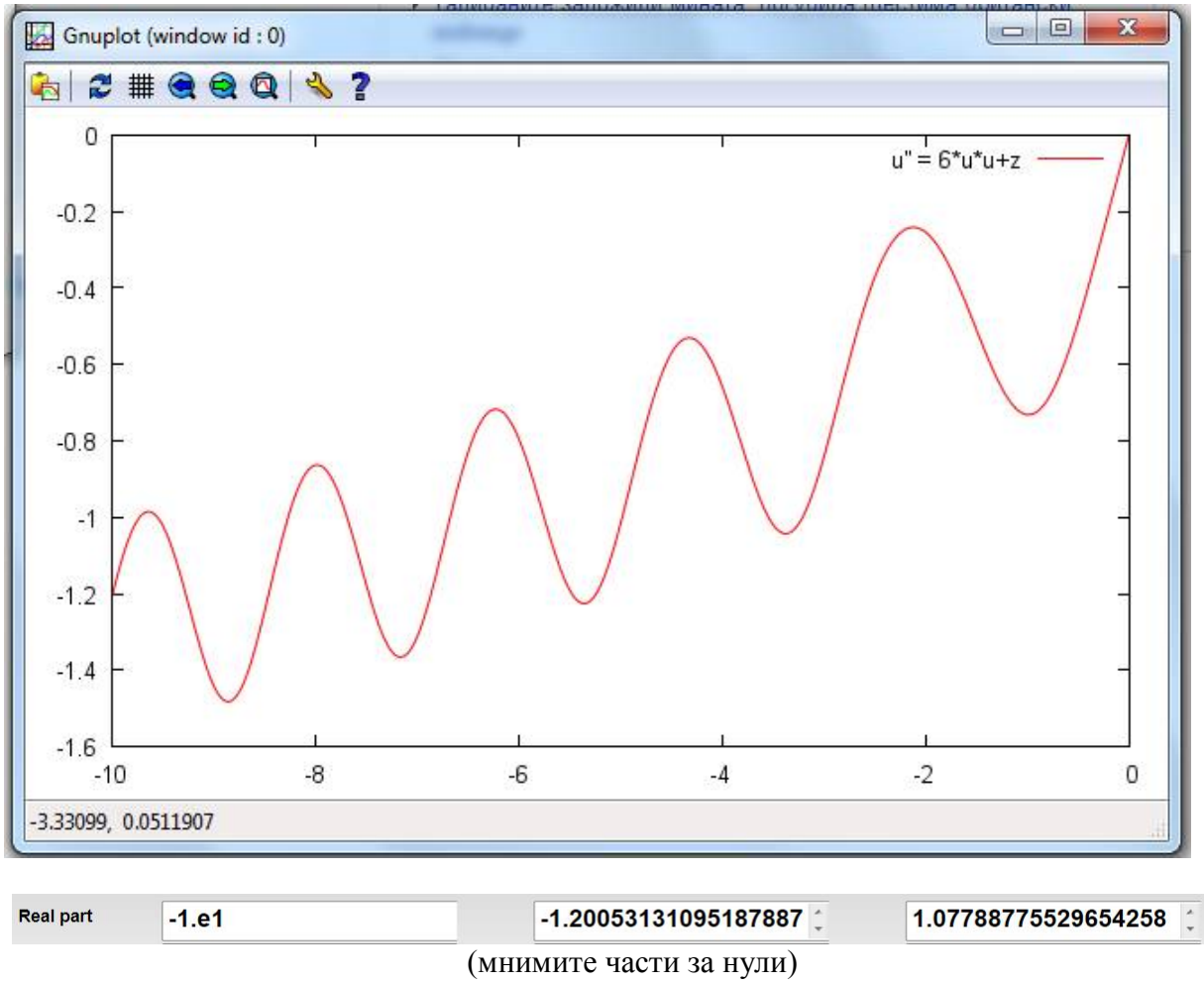


Фигура 3.4. Задаване на стойност на параметър.

Всички останали възможности за промяна на задачата се блокират до въвеждане на валидни стойности в таблицата за параметри. Въвеждането на параметри завършва с потвърждение (“Set params”). Стойностите по подразбиране в таблицата с параметрите са нули. За реалната и мнимата част на всеки параметър се допускат само валидни записи на реални числа. Ако след потвърждението е открит невалиден запис, фокусът се премества в клетката с грешка. Процесът на въвеждане на параметри може да завърши само с валидни числа.

Примери

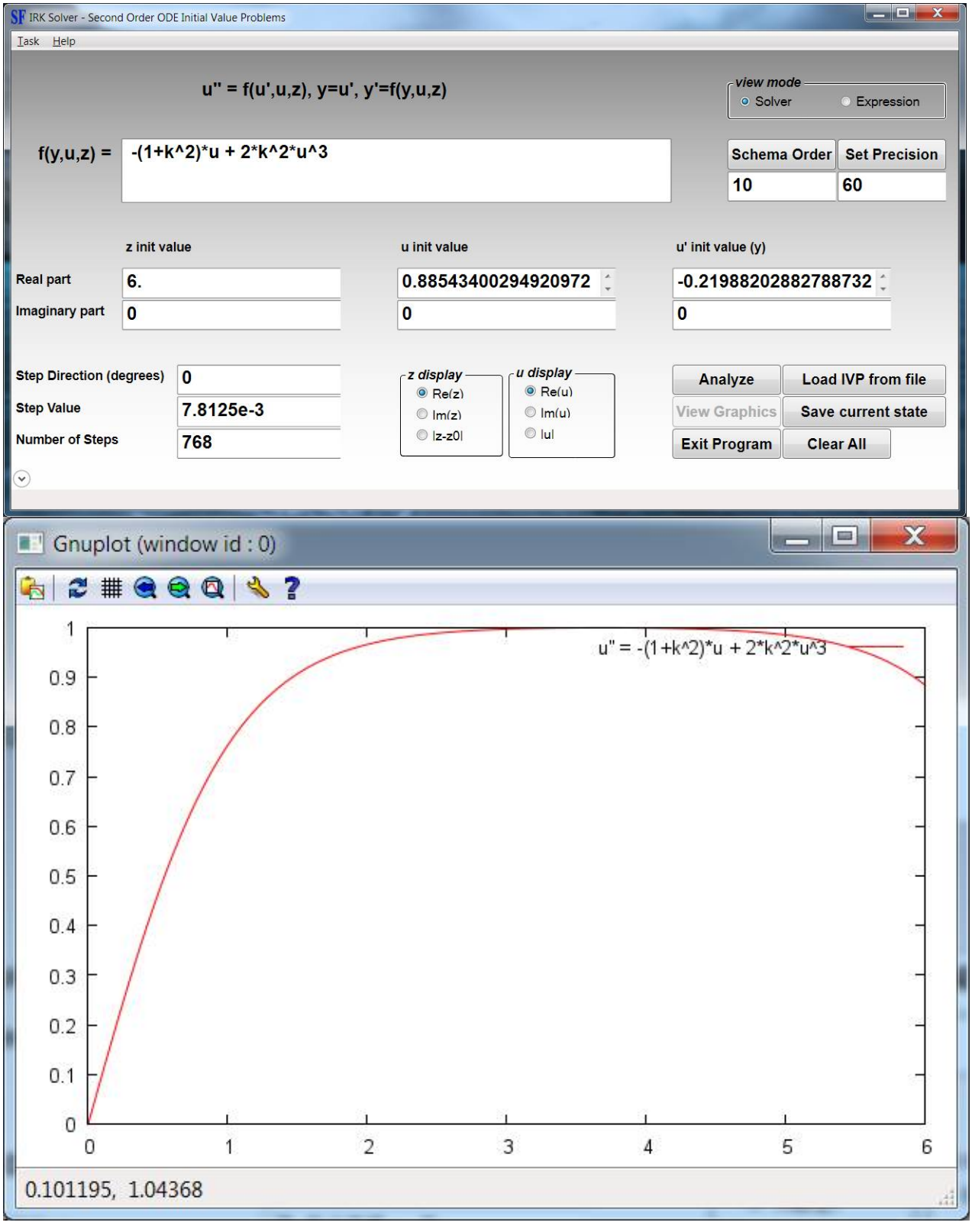
Уравнението $u'' = 6u^2 + z$ (уравнението P_1 или първото уравнение на Пенлеве) с 1280 стъпки в отрицателна посока с големина на стъпката по подразбиране, но със знак минус (от 0 до -10) и всички други стойности по подразбиране дава следния резултат.



(мнимите части за нули)

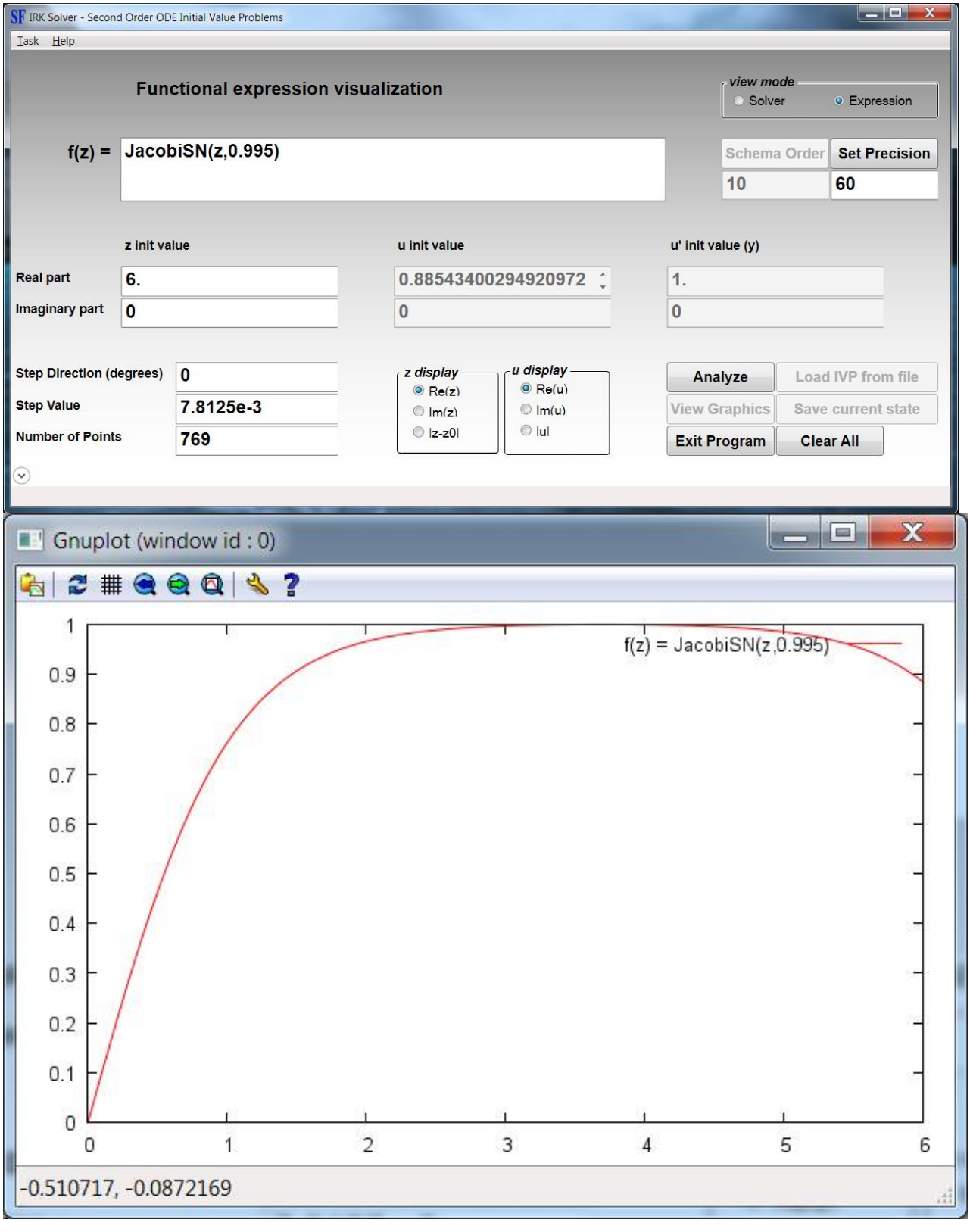
Фигура 3.5. Резултат от пресмятането на $u'' = 6u^2 + z$ със зададените в текста начални условия.

Следващият пример е тест с $u'' = -(1+k^2)u + 2k^2u^3$. Това е диференциалното уравнение за функцията на Якоби sn(z|k). Записваме в полето за уравнението -(1+k^2)*u + 2*k^2*u^3. Задаваме 768 стъпки (пресмятане от 0 до 6) и всички други начални стойности по подразбиране. За стойности на *k* близки до 1, кривата на решението има характерно плато. Задаваме на параметъра *k* стойност 0.995 за (нула за мнимата част).



Фигура 3.6. Резултат от пресмятането на $u'' = -(1+k^2)u + 2k^2u^3$ със зададените в текста начални условия.

Един възможен вид проверка е визуализиране на самото решение в режим на израз. Броят на точките е с единица по-голям от броя на стъпките в предишния пример:



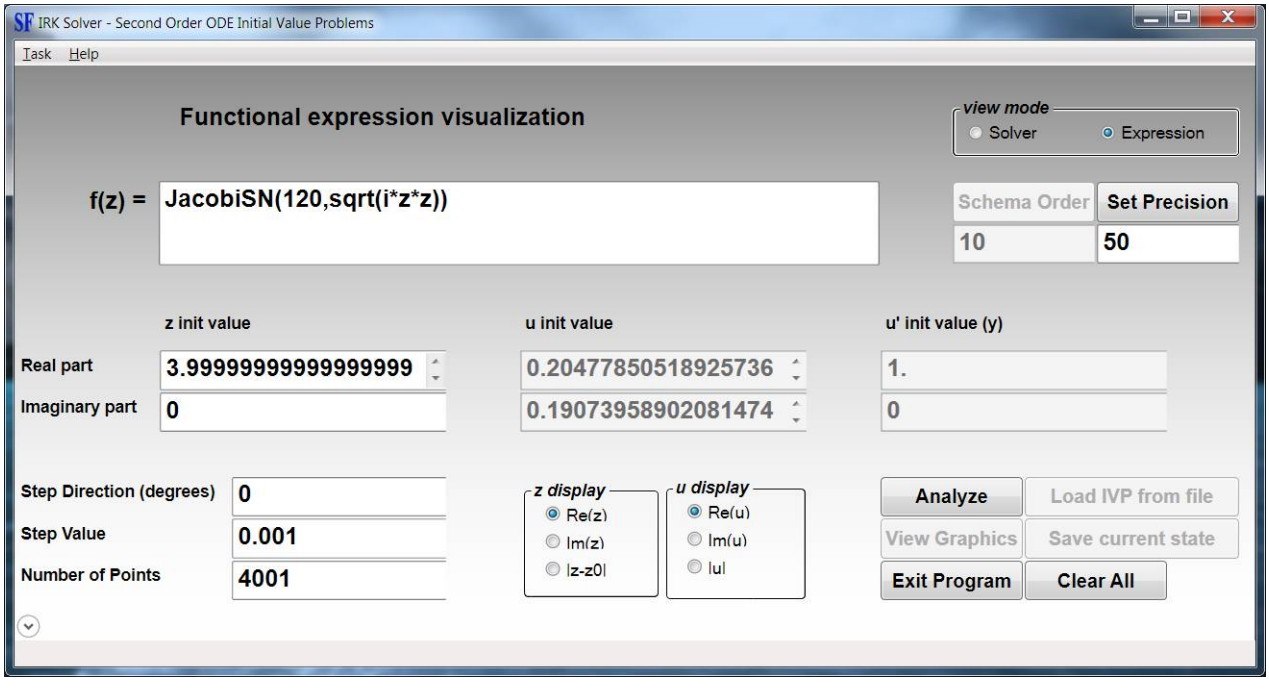
Фигура 3.7. Резултат от визуализацията на sn(z,0.995) - решение на предишната задача за пресмятането на $u'' = -(1+k^2)u + 2k^2u^3$ със зададените в текста начални условия.

Графиките изглеждат идентични, ако не е напомнянето в легендата по какъв начин са получени. В полето за получената последно стойност (под ”u init value”) също няма разлика във видимата част. Тази стойност се проверява с SFCALC (идентична с тази, получена при визуализиране на функционален израз). Точните стойности в съответните полета (трябва да се превъртят, освен това, в случая на визуализация на израз трябва да се мине в режим решател, за да бъде достъпно полето) са съответно

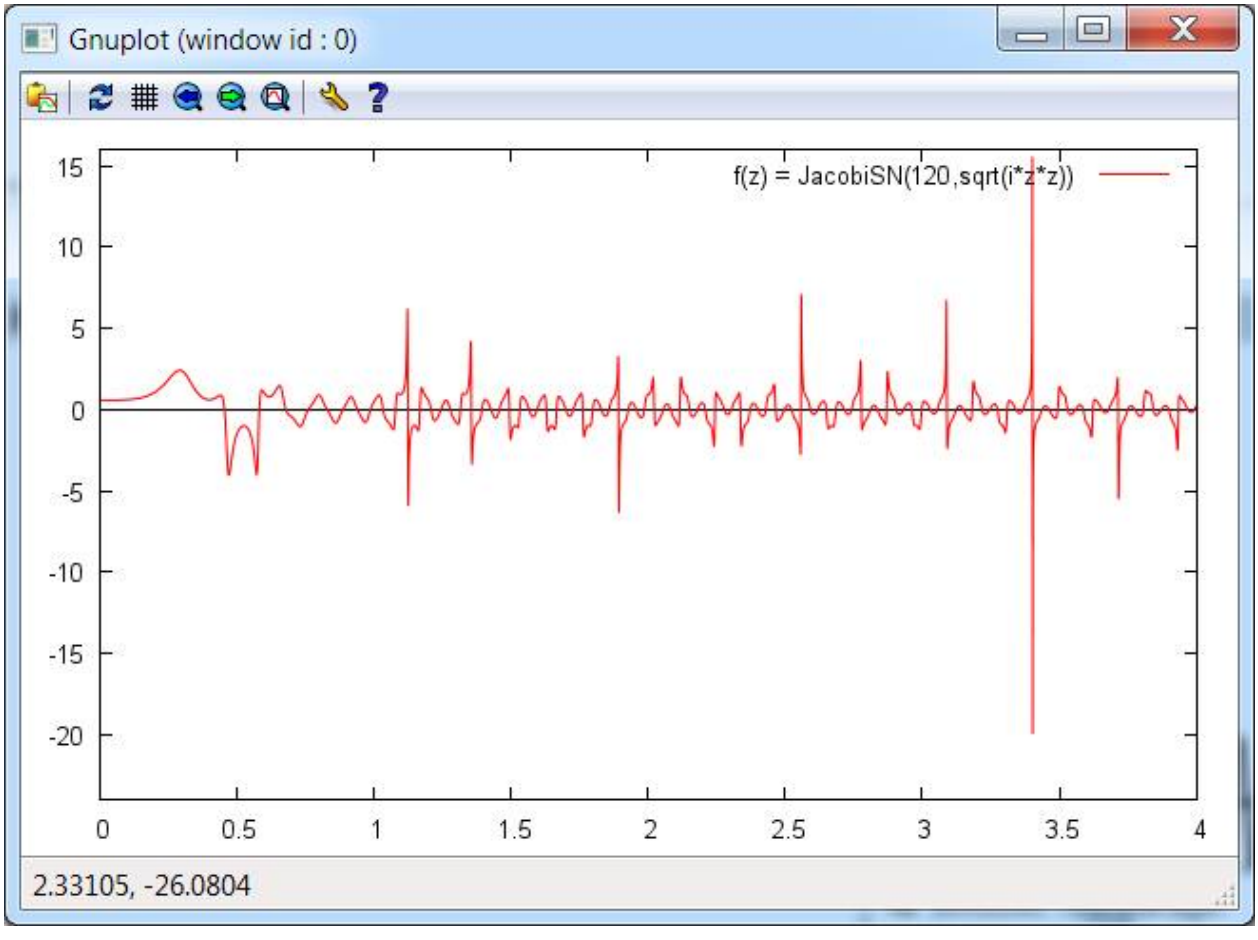
0.885434002949209727996777289178042343651725980734510419723528,

0.885434002949209727996777289178042343651725980734510419723528.

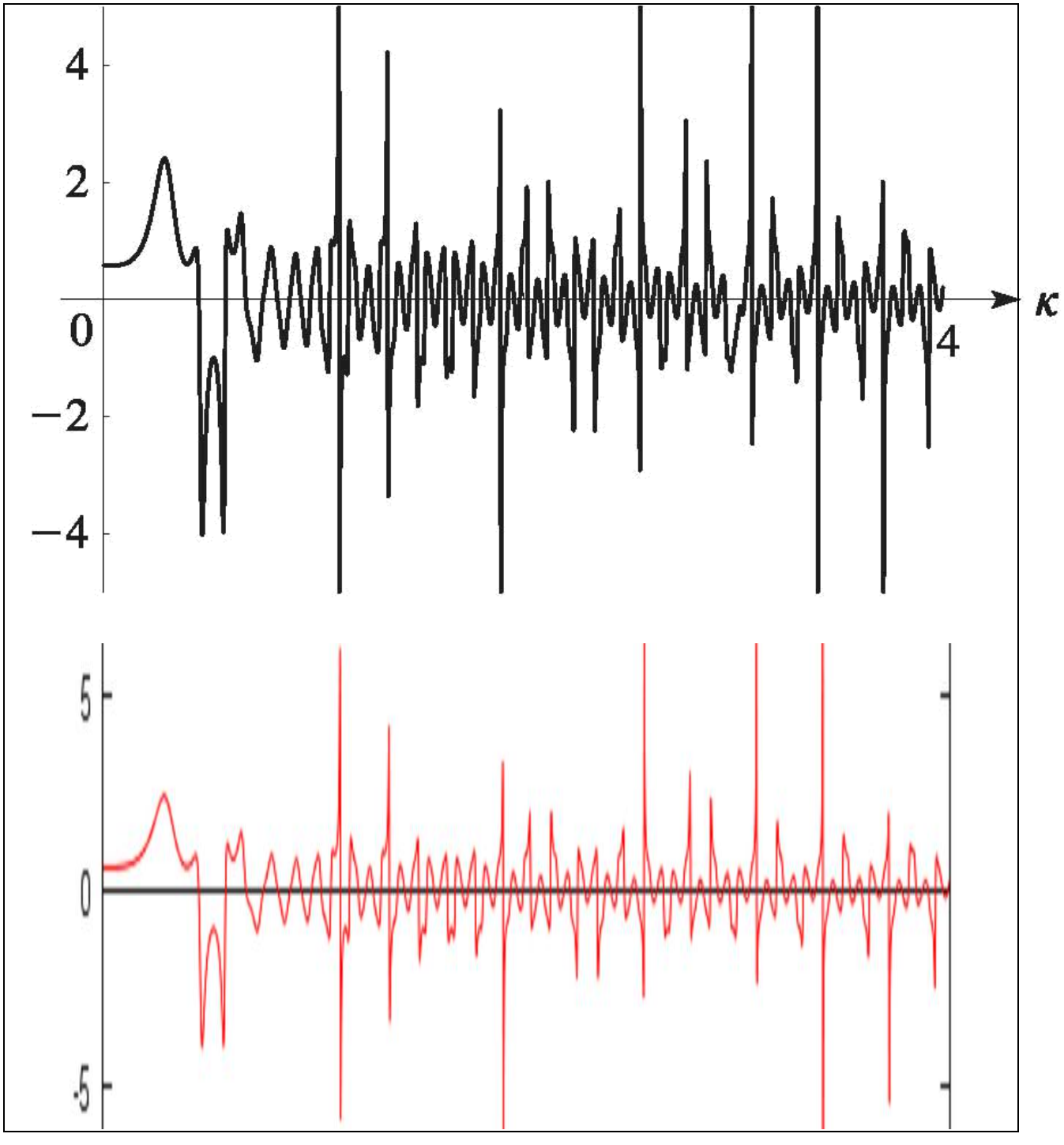
И един пример за проследяване в комплексната област. Примерът е за sn(120,k), където $k^2 = i\kappa^2$ в интервала $\kappa \in [0,4]$. Имагинерната единица се задава като параметър с реална и мнима част 0, 1. Примерът не е избран случайно. Графиката за него има в [106] стр. 552 където има печатна грешка в зависимостта на параметъра - записана е като $k^2 = i\kappa$ - в сайта на книгата [103] няма грешка). Този пример беше използван за тест, който в първия момент изглеждаше неуспешен, поради различния мащаб (графиката в [106] е и отрязана в по-тясна ивица, приблизително ± 5 около хоризонталната ос). Съответната графика за мнимата част на sn в същия диапазон се получава чрез избиране на "Im(u)" в полето "*u display*" (и също е отрязана малко в [106]).



Фигура 3.8. Задаване на визуализацията на реалната част на sn при фиксирана стойност на първия аргумент и зависимост от втория.



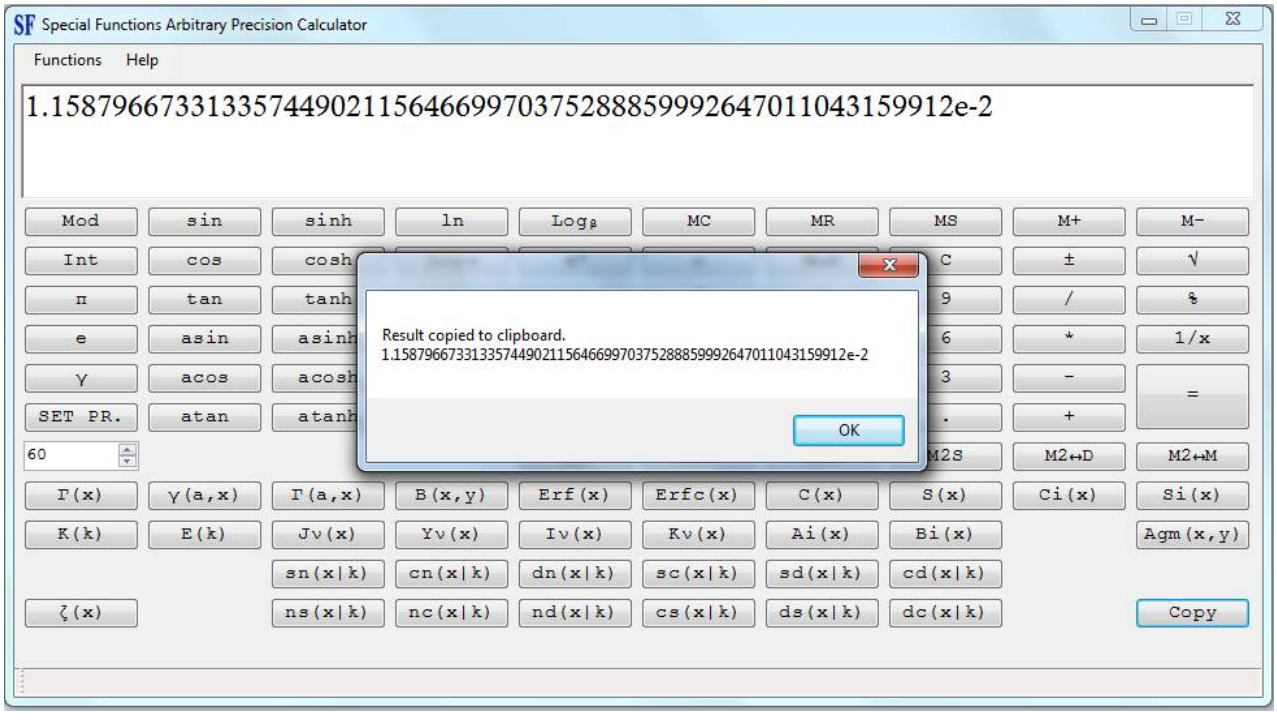
Фигура 3.9. Получената визуализация на реалната част на sn.



Фигура 3.10. Сравнение с графиката [106] (в горната част), след подходящо мащабиране.

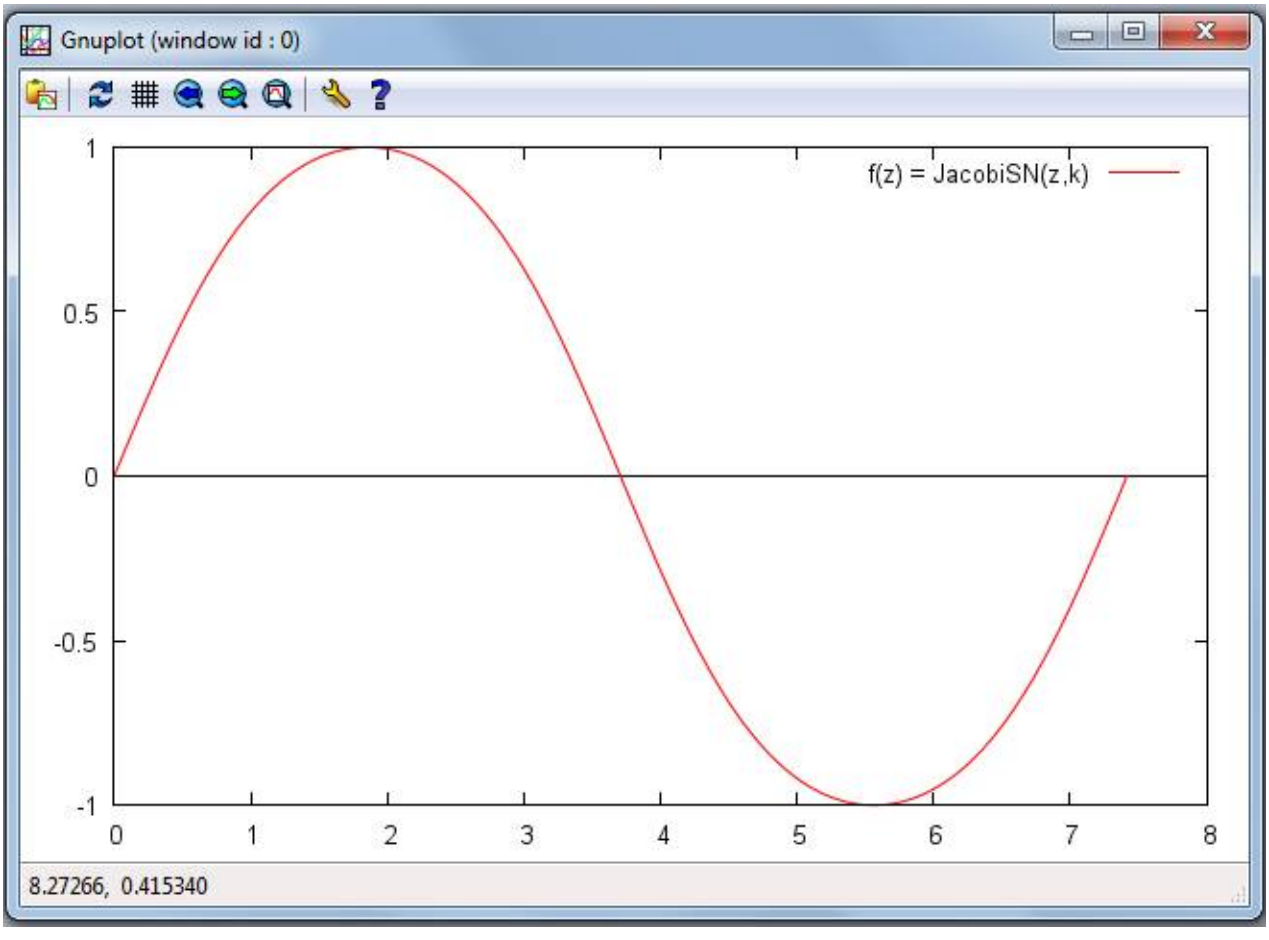
Следва пример за прехвърляне на необходимите константи. Да речем, че ни трябва да визуализираме sn(z|k) с аргумент $k = 1/\sqrt{2}$ за един период от начална точка 0 и 640 подинтервала (641 точки на кривата). Периодът е $4K\left(1/\sqrt{2}\right)$, където K е пълен елиптичен интеграл от първи род, така че необходимата стъпка е $4K\left(1/\sqrt{2}\right)/640$. Ще въведем sn(z|k) като функционален израз с параметър k. Превключваме в режим на израз и въвеждаме

JacobiSN(z,k) за израза. Въвеждаме 641 за броя на точките. След това отваряме SFCALC (ако не е вече отворен). Въвеждаме 60 в полето под бутона ”SET PR.” и потвърждаваме (за да имаме същата точност, иначе точността по подразбиране за SFCALC е 120 десетични знака). Набираме последователно бутоните 2, $\sqrt{}$, 1/x, $K(k)$, *, 4, /, 6,4,0, = и получаваме резултата. След това пренасяме резултата в клипборда (бутон Copy).



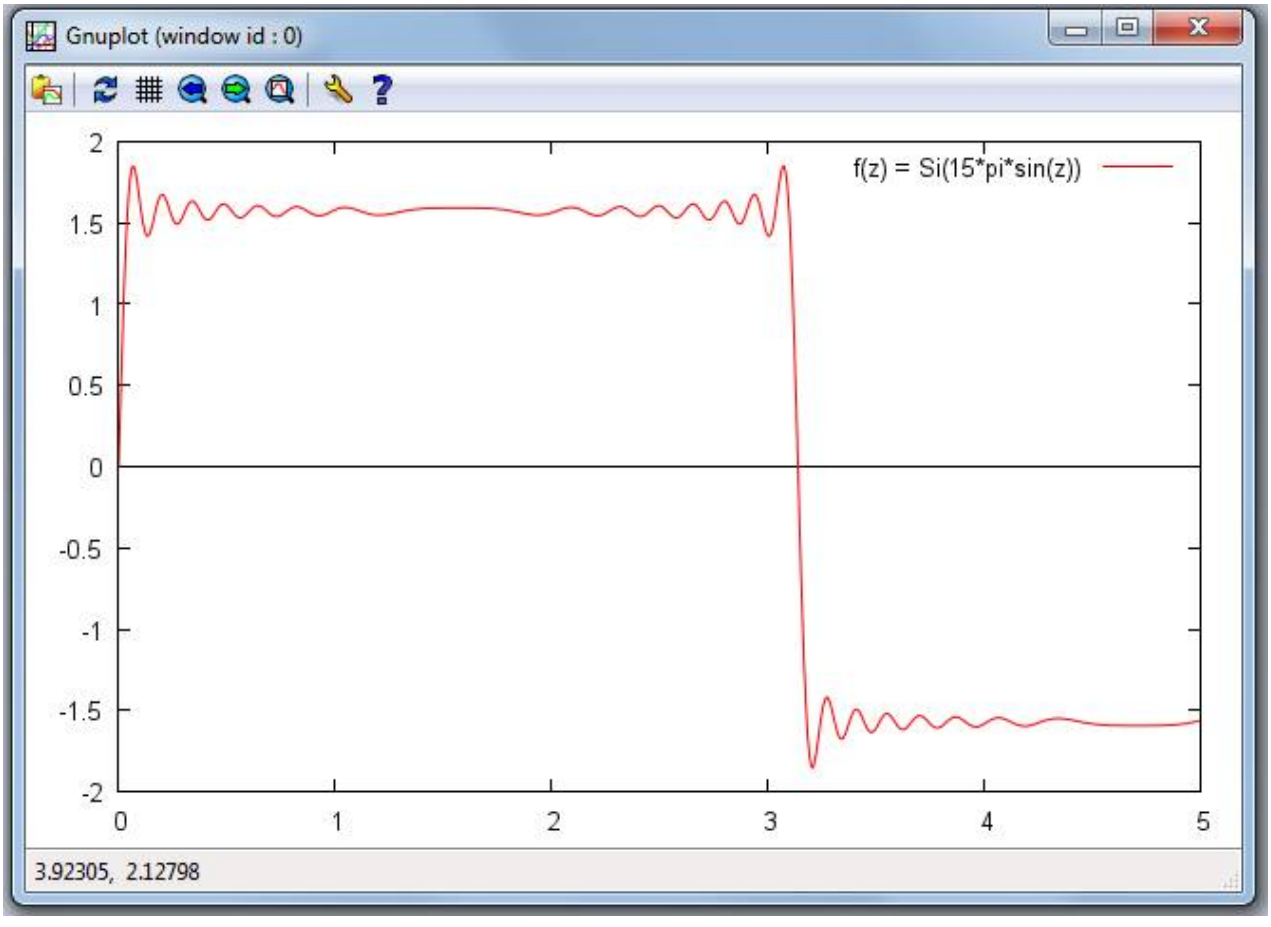
Фигура 3.11. Копиране на резултат от SFCALC в клипборда.

Потвърждаваме с OK и преминаваме в VODE2. Въвеждаме нужната стъпка в полето ”Step Value” с Ctrl+Ins. Бутонът Analyze отваря таблицата за параметрите. Преминаваме отново към SFCALC и пресмятаме необходимата стойност $1/\sqrt{2}$ (2, $\sqrt{}$, 1/x), след което копираме тази стойност в клипборда (бутон Copy). В VODE2 изчистваме полето за реалната част на параметъра k и копираме (Ctrl+V). Завършваме процеса на въвеждане с ”Set params”. След избиране на бутона “View Graphics”, който сега е разрешен, имаме желания един период:



Фигура 3.12. Полученият един период на $\mathrm{sn}\left(z,1/\sqrt{2}\right)$.

Накрая още един пример за визуализиране на функционален израз. Искаме да видим как изглежда графиката на $\mathrm{Si}(15\pi\sin(z))$. Тук Si е интегралният синус. За интервала [0,5] графиката е приведена по-долу.



Фигура 3.13. Графиката на $\mathrm{Si}(15\pi \sin(z))$ в интервала [0,5].

Някои особености на VODE2

Към момента този програмен инструмент не е оптимизиран за разход на памет. "Числата с произволна точност" са обекти от тип препратка (reference objects) и се създават интензивно (в свободната памет, heap) от някои процедури в този случай. Това е така, защото класът на комплексните числа предефинира някои обичайни аритметични операции, което от своя страна изисква създаването на нов обект в лявата част на изрази от типа "x = b + c" и подобни. Това може да се проследи от Windows Task Manager (раздел Performance на Task Manager). Проблемът е принципно решим (създавайки например специален пул с работни променливи, които да се използват по специален начин за многократна употреба), но това не е направено засега в тази програма, която от друга страна демонстрира възможности за обединяване на различни възможности (решаване в комплексната област и разбира се специфичен интерфейс) и беше нещо като изпитателен полигон в програмната система. По-вероятно е да се разработи интерфейс на решателя в общия случай (произволна размерност), но с реални променливи. Режимът за визуализация на функционален израз си остава изключително полезен и в по-широк контекст на други приложения.

3.4. Заключение.

Разработени са програмни инструменти за пресмятане на (системи) обикновени диференциални уравнения с много висока точност. Един от тях е специализиран (VODE2) и демонстрира възможностите за създаване на интерактивен графичен програмен инструмент в използваната среда за важен частен случай (уравнение от втори ред). Два други (IRK_Test, ODEX) дават възможност за третиране на обща задача за решаване на система ОДУ. Първият от тях дава някои предимства при по-продължителни симулации за някои класове задачи, а вторият е с акцент върху ефективността.

Глава 4. Методи за подобряване на намирането на корени на скаларни уравнения с висока точност.

4.1. Методи с локална сходимост.

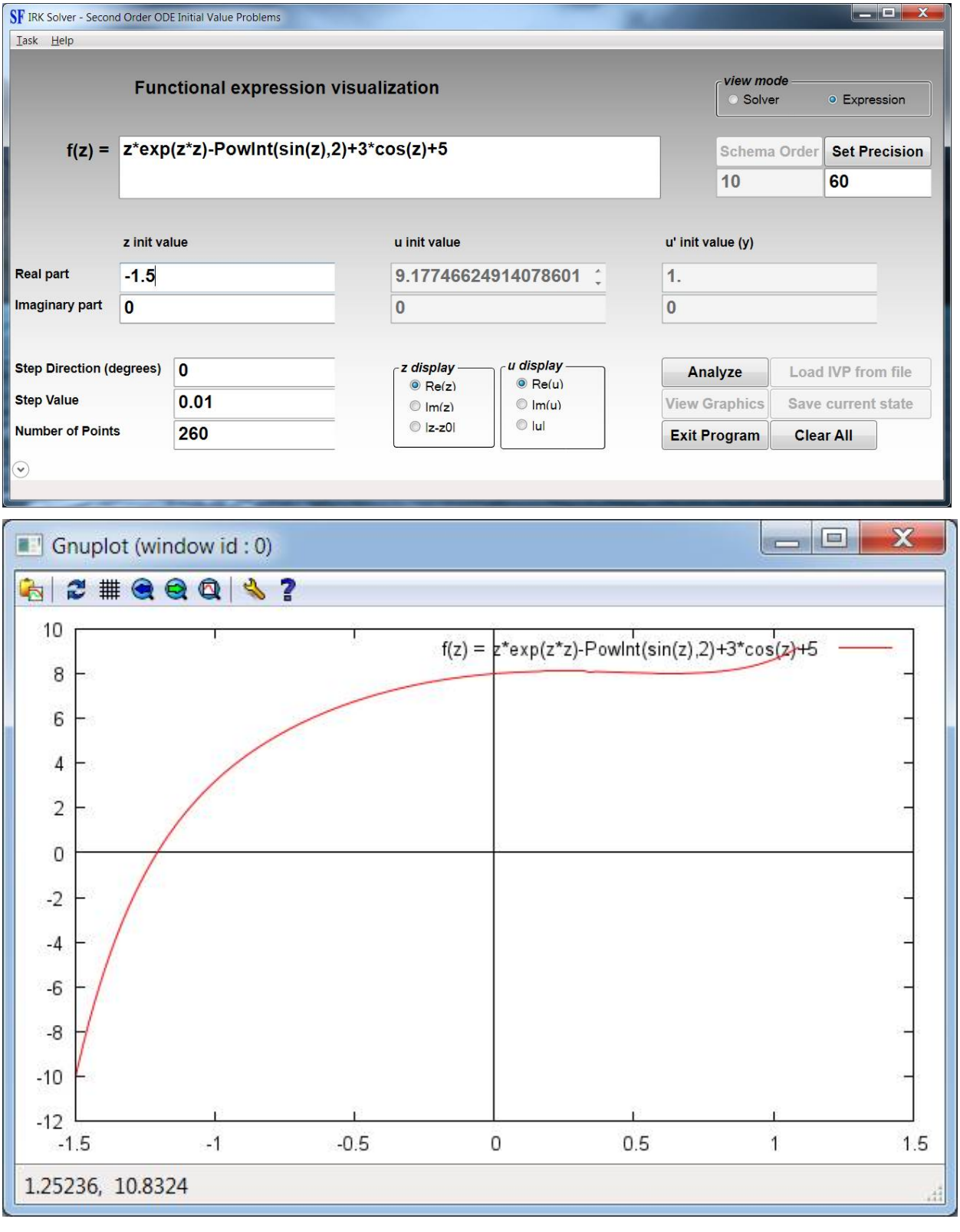
Класификацията на методите може да се прави по-различни критерии: скорост на сходимост, изискване за информация в предишни приближения, изчислителна ефективност и др. Източник на такива определения, както и разбира се на описанието на множество интересни методи са [138], [109]. Заслужава специално отбелязване и текстът на дисертационния труд [124], който предизвика създаване на програмен инструмент, който освен другото може да служи за тестване и сравняване на вградените методи. Като начало бе използван много добре оформеният набор от тестови функции там, някои от които нетривиални примери от реално възникнали на практика задачи в различни области.

Предлаганият програмен инструмент MPRootFindTestLocal, позволява въвеждане от командния ред на самата функция, началното приближение, използваната точност на пресмятанията и метода за решаване, например

```
MPRootFindTestLocal "x*exp(x*x)-PowInt(sin(x),2)+3*cos(x)+5" -1 200 M8_1
```

Това е функция 7, на таблица 2.1 в [124]. Кавичките е добре да се слагат, макар и да не са задължителни, ако в израза за функцията няма интервали. 200 е точността на пресмятанията в десетични знаци. -1 е началното приближение, а M8_1 е името на метода (4.5.15), раздел 4.5 "Оптимални методи от осми ред" в [124] с $\alpha = 1$. Освен резултата -1.2076... с грешка 2.4043e-212, се получава и известна допълнителна информация, 4 итерации, 12 пресмятания на функцията и 4 пресмятания на производната. Решаването на същото уравнение с класическия метод на Островски дава 5 итерации, 9 пресмятания на функцията и 4 пресмятания на производната. За метода на Нютон тези стойности са съответно 10, 10, 9.

Тук, разбира се, се решава тестов пример с предварително зададено начално приближение, но в реална ситуация, поради важността на началното приближение, е добре да имаме средство за визуализация. Едно такова е предвидено в програмната система, като специален режим на визуализация на програмния инструмент VODE2 за решаване на ОДУ от втори ред. В този режим (expression) се визуализира функция. Визуализация за функцията от горния пример с начална точка -1.5, и 260 точки с разстояние между тях 0.01 (интервал [-1.5, 1.1]) изглежда така.



Фигури 4.1. Задаване на визуализация на функция в интервал в специалния режим "Expression" на VODE2 и самата визуализация в gnuplot.

Програмният инструмент е конструиран така, че съдържа използваните методи в отделен файл, за да могат лесно да се добавят нови. Освен това управляващата го програма

е проектирана да получава като аргументи функцията и производната, така че има известна гъвкавост, ако трябва да се използва без интерпретатор и символно диференциране, а със зададени в извикващия код функции. Това се прави например при пресмятане на някои константи (както е описано за константата на Раманудджан-Солднер и границата на Лаплас в главата за пресмятане на математически константи).

4.2. Опит за преодоляване на проблема с локалната сходимост. Вариация на метод на продължението чрез хомотопия.

Функциите за които прилагането на локален метод с произволно начално приближение дава решение са твърде малко (изпъкнали, унимодални). В случай, че е известен интервал, в който се намира коренът, може разбира се да се приложи метода на разполовяването (bisection). Това обаче не е много ефективно и на практика този метод винаги се прилага в комбинация с някакъв бърз локален метод, след достигане на околност, в която последният е сходим. Освен това методът на разполовяването не може да се обобщи по очевиден начин за по-голяма размерност. Тук директно ще се опишат идеите, използвани за реализирания в програмната система вариант на метод на продължението чрез хомотопия (Homotopy Continuation Method, [48], [81]). Освен несъмнената си елегантност, той има предимството за лесно обобщаване за многомерни задачи и освен това може да се видоизмени за проследяване на много решения [112].

Методът е в групата на така наречените методи на продължението (Numerical Continuation Methods) [4]. Основната им идея е в последователно решаване на уравнения от вида $F(x,\lambda)=0$ за поредица стойности на числовия параметър $\lambda=\lambda_0,\lambda_1,\ldots,\lambda_n$, всяко от които дава подходящо начално приближение за следващото, като $F(x,\lambda_n)=f(x)=0$ съвпада с първоначалната ни задача. Формално хомотопията на две непрекъснати функции $f,g:\mathbb{R}\to\mathbb{R}$ е непрекъсната функция $H:\mathbb{R}\times[0,1]\to\mathbb{R}$, за която $H(x,0)=f(x)$ и $H(x,1)=g(x)$. При продължение чрез хомотопия, най-често използваните варианти на хомотопия са така наречената нютонова хомотопия и хомотопия на неподвижната точка, определени съответно с

(4.1) $$H(x,t)=f(x)-(1-t)f(x_0)$$

и

(4.2) $$H(x,t)=(1-t)(x-x_0)+tf(x),$$

за някакво x_0 (начално приближение). В дадените случаи поредицата от стойности $t=t_0=0,\ldots,t_n=1$ започват при t = 0 от функцията $f(x)-f(x_0)$ (нютонова хомотопия) или x - x_0 (хомотопия на неподвижната точка) и завършват при t = 1 с първоначалната функция $f(x)$.

Идеята е да се следва параметризираната крива на решенията, получена от диференцирането на

(4.3) $$H\big(t(s),x(s)\big)=0$$

от t = 0 до t = 1. Това е всъщност задача за решаване на система ОДУ с начални условия

$$(4.4)\qquad \begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}s} = -\dfrac{\partial H}{\partial t}, \\ \dfrac{\mathrm{d}t}{\mathrm{d}s} = \dfrac{\partial H}{\partial x}, \\ \\ (x,t)\big|_{s=0} = (x_0, 0) \end{cases}$$

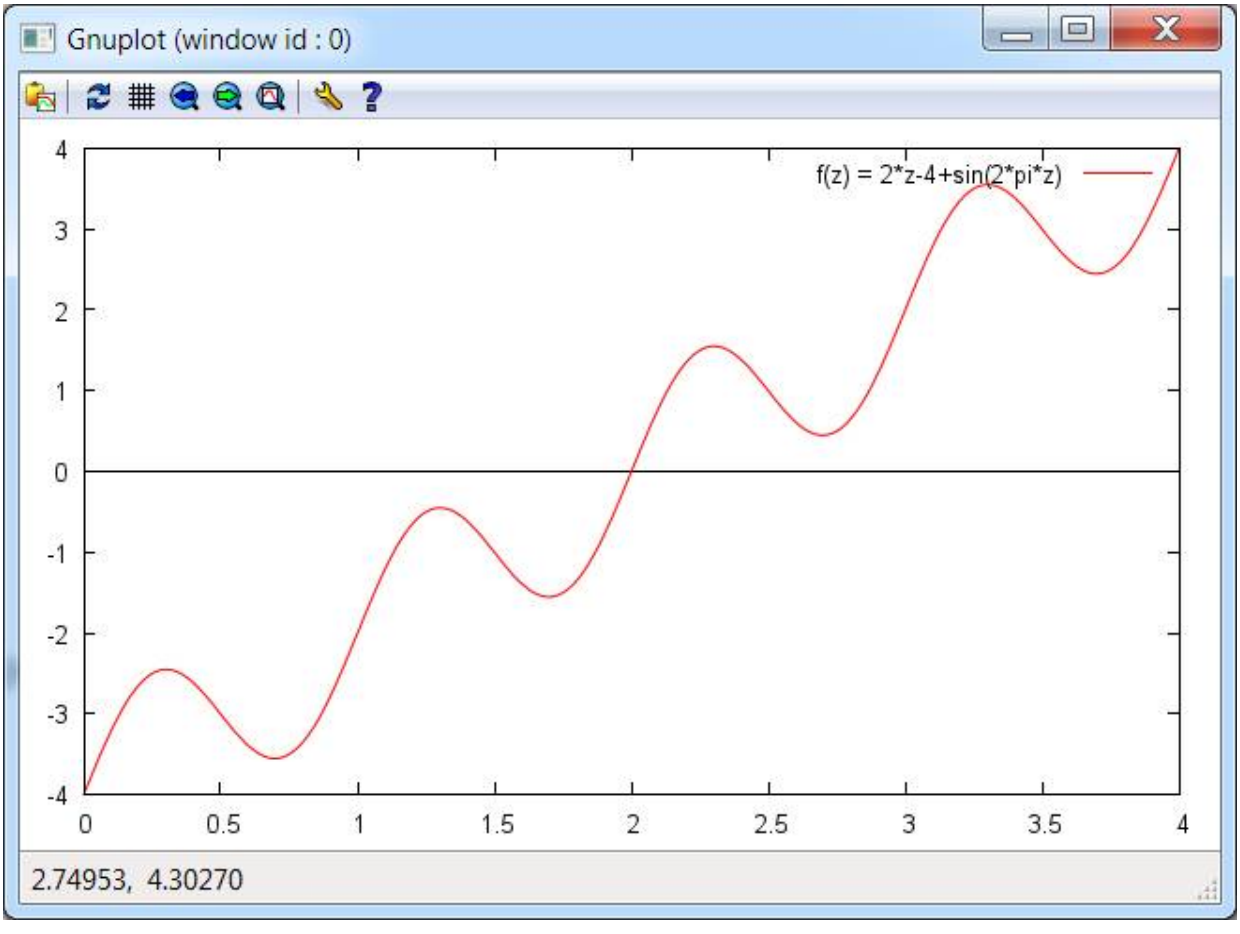
до достигането $t(s)$=1. (В [81] е дадено едно ясно описание на възможните сценарии в общия случай, включващи точки на обръщане и точки на бифуркация.)

Получената по този начин стойност $x(1)$ може да се използва след това като начална точка на бързо сходящ локален метод, за получаване на желана точност (уточняване на решението). В програмната система разполагаме с подходящо средство за решаване на (4.4) във вида на публичния клас IRK_Solver, проектиран за решаване на системи ОДУ чрез използване на неявна схема на Рунге-Кута от произволен ред. Резултат от комбинирането на изложените по-горе идеи плюс реализираните вече от нас инструментални средства дадоха възможност за реализацията на програмния инструмент MPRootFindTestHN. От командния ред освен израз за функцията, изисквани параметри са начална точка, желана точност и ред на неявната схема на Рунге-Кута.

Нека разгледаме функцията (примерът е от [81])

$$f(x) = 2x - 4 + \sin(2\pi x).$$

Макар и проста, графиката ѝ заслужава да се покаже, тъй като дава представа защо локалните методи не се справят с намиране на корена (в случая е x = 2, единствен, с кратност единица), ако началната точка не е близка до корена. По принцип те генерират всяко следващо приближение в някаква посока, произтичащо от някакъв (неявен) приближен модел на функцията (различен за различните методи) *в околност на текущата точка*.



Фигура 4.2. Пример на типична функция, затрудняваща локалните методи, но решавана с лекота от метода на продължението чрез хомотопия.

При начална точка -8 и изисквана точност 100 десетични знака и за трите метода, споменати при разглеждането на MPRootFindTestLocal в предишния раздел, липсва сходимост. При тези условия и избор на ред 8 за неявна схема на Рунге-Кута, достигната от MPRootFindTestHN точка дори не се нуждае от доуточняване (при крайното подаване на уточняващия метод, вече има необходимата точност). Това разбира се, не е общо правило, а следствие на предложената реализация, в която след стъпката, надхвърляща $t = 1$, тя се повтаря няколко пъти, намалявайки стъпката на IRK_Solver два пъти при всяко повторение до стойност, при която t не надхвърля единица. Често това дава желаната точност и без по-нататъшно уточняване.

4.3. Намиране на корените на някои специални функции с висока точност.

В този раздел ще се разгледат въпросите, свързани с намирането на корени на класически ортогонални полиноми, функции на Бесел и функции на Ейри.

4.3.1. Намиране на корените на класически ортогонални полиноми

Основните източници на информация за този подраздел са [134], [95]. [139], [96].

Методът ADK (Aberth, Durand, Kerner) е особено подходящ, тъй като в дадения случай всички корени са реални и прости (кратност единица). Методът ADK е модификация на метода на Нютон с автоматично намаляване на степента (implicit deflating). Ако $x^{(1)}, x^{(2)}, \cdots, x^{(r)}$ са изчислените до момента корени на полинома $p(x)$, то итеративната формула

$$x_{n+1} = x_n - \frac{p(x_n)}{p'(x_n) - p(x_n)\sum_{j=1}^{r}\frac{1}{x_n - x^{(j)}}} \tag{4.5}$$

е сходяща към друг корен на полинома.

Стратегията се състои в избиране на първоначално приближение $x = R$, което е по-голямо от най-големия корен на полинома и последователно намиране всички корени с итеративната формула ADK (използва се разбира се и евентуалната четност/нечетност на полинома).

Стойността на R за съответния полином може да се вземе от следващата таблица.

Полином на	*R*
Якоби	1
Лежандър	1
Лагер	$2n + \alpha - 2 + \sqrt{1 + 4(n-1)(n+\alpha-1)}$
Ермит	$\sqrt{2n+1}$

Таблица 4.1. Начално приближение за намиране на следващ корен в ортогонални полиноми.

От важните частни случаи на полиномите на Якоби $P_n^{(\alpha,\beta)}(x)$ - ултрасферичните полиноми $C_n^{(\lambda)}(x) = P_n^{\left(\lambda-\frac{1}{2},\lambda-\frac{1}{2}\right)}(x)$, наричани също полиноми на Гегенбауер, отделна реализация има за полиномите на Лежандър $P_n(x) = C_n^{\left(\frac{1}{2},\frac{1}{2}\right)}(x) = P_n^{(0,0)}(x)$. Полиномите на Чебишев от първи и втори вид трябва да се пресмятат направо чрез представянето си като частен случай на полиномите на Якоби (съответно $T_n(x) = C_n^{(0)}(x) = P_n^{\left(-\frac{1}{2},-\frac{1}{2}\right)}(x)$ и $U_n(x) = C_n^{(1)}(x) = P_n^{\left(\frac{1}{2},\frac{1}{2}\right)}(x)$).

Ако се търсят корените на полином от висока степен с голяма точност, целесъобразно е първоначално да се намерят с помощта на ADK с по-малка точност (примерно с половината от исканите десетични знаци) и след това да се "доуточнят" с по-бърз метод. При нас в ролята на доуточняващ метод се използва от метода на Лагер (вж. приложението в края на този подраздел). В съответствие с това за полиномите на Якоби, Лагер и Ермит (съответно $P_n^{(\alpha,\beta)}(x)$, $L_n^{(\alpha)}(x)$ и $H_n(x)$) и техни първи и втори производни са реализирани няколко процедури. Вторите производни са необходими за "доуточнителя".

За пресмятането се използва самите полиноми се използва рекурентната зависимост

$$p_{n+1}(x) = \left(A_n x + B_n\right) p_n(x) - C_n p_{n-1}(x). \tag{4.6}$$

където за полиномите на Якоби константите са

$$A_n = \frac{\left(2n+\alpha+\beta+1\right)\left(2n+\alpha+\beta+2\right)}{2\left(n+1\right)\left(n+\alpha+\beta+1\right)}, \tag{4.7}$$

$$B_n = \frac{\left(\alpha^2-\beta^2\right)\left(2n+\alpha+\beta+1\right)}{2\left(n+1\right)\left(n+\alpha+\beta+1\right)\left(2n+\alpha+\beta\right)}, \tag{4.8}$$

$$C_n = \frac{\left(n+\alpha\right)\left(n+\beta\right)\left(2n+\alpha+\beta+2\right)}{\left(n+1\right)\left(n+\alpha+\beta+1\right)\left(2n+\alpha+\beta\right)}. \tag{4.9}$$

Съответно за полиномите на Лежандър, Лагер и Ермит тези константи са в таблицата по-долу.

	A_n	B_n	C_n
$P_n(x)$	$\frac{2n+1}{n+1}$	0	$\frac{n}{n+1}$
$L_n^{(\alpha)}(x)$	$-\frac{1}{n+1}$	$\frac{2n+\alpha+1}{n+1}$	$\frac{n+\alpha}{n+1}$
$H_n(x)$	2	0	$2n$

Таблица 4.2. Коефициенти в рекурентната зависимост на ортогоналните полиноми.

За производните се използват следните рекурентни зависимости

$$\begin{aligned}&\left(2n+\alpha+\beta\right)\left(1-x^2\right)\frac{\mathrm{d}}{\mathrm{d}x}P_n^{(\alpha,\beta)}(x) = \\ &= n\left(\alpha-\beta-\left(2n+\alpha+\beta\right)x\right)P_n^{(\alpha,\beta)}(x) + 2\left(n+\alpha\right)\left(n+\beta\right)P_{n-1}^{(\alpha,\beta)}(x)\end{aligned}, \tag{4.10}$$

$$\left(1-x^2\right)\frac{\mathrm{d}}{\mathrm{d}x}P_n(x) = -2xP_n(x) + 2P_{n-1}(x), \tag{4.11}$$

$$\frac{\mathrm{d}}{\mathrm{d}x}L_n^{(\alpha)}(x) = -L_{n-1}^{(\alpha+1)}(x), \tag{4.12}$$

$$\frac{\mathrm{d}}{\mathrm{d}x}H_n(x) = 2nH_{n-1}(x). \tag{4.13}$$

Вторите производни се определят от диференциалното уравнение за дадения полином. Общият вид на уравнението е

$$A(x)f''(x) + B(x)f'(x) + \lambda_n f(x) = 0, \tag{4.14}$$

а съответните коефициенти са приведени в таблицата

	$A(x)$	$B(x)$	λ_n
$P_n^{(\alpha,\beta)}(x)$	$1-x^2$	$\beta-\alpha-(\alpha+\beta+2)x$	$n(n+\alpha+\beta+1)$
$P_n(x)$	$1-x^2$	$-2x$	$n(n+1)$
$L_n^{(\alpha)}(x)$	x	$\alpha+1-x$	n
$H_n(x)$	1	$-2x$	$2n$

Таблица 4.3. Таблица за коефициенти в рекурентната зависимост за вторите производни на ортогоналните полиноми.

Като част от процедурата за тестване бяха пресметнати съответните полиноми от степен 25 000 с точност 200 знака, заедно с теглата на съответните им квадратурни формули от гаусов тип, чиято сума накрая служи за проверка. За най-тежкия случай на полином на Якоби (от общ вид) на по-стара машина с процесор Intel Core 2 Duo E8200 времето е 14 часа. На по-нов преносим компютър с процесор Intel Core i7-3610QM, времето е 7 часа.

Това е със старите файлове на динамичните библиотеки, получавани в варианта xmpir от 2010 година. За новите динамични библиотечни файлове, които освен това са настроени за конкретния процесор, тестове не са проведени все още.

Тук посочваме възможна точка на растеж. Евентуална реализация на метод с едновременно пресмятане на корените, което би позволило паралелна обработка.

Приложение. Метод на Лагер.

Методът на Лагер използва итеративната формула

$$x_{k+1} = x_k - \frac{n}{\frac{p'(x_k)}{p(x_k)} \pm \sqrt{(n-1)\left[(n-1)\frac{p'(x_k)^2}{p(x_k)^2} - n\frac{p''(x_k)}{p(x_k)}\right]}}, \tag{4.15}$$

където n е степента на полинома, а знакът в знаменателят е като този на $p'(x_k)$. Ако положим съответно $G_k = \frac{p'(x_k)}{p(x_k)}$ и $H_k = G_k^{\,2} - \frac{p''(x_k)}{p(x_k)}$, то тази формула се записва по компактно като

$$x_{k+1} = x_k - a_k\,, \quad a_k = \frac{n}{G_k \pm \sqrt{(n-1)(nH_k - G_k^{\,2})}}\,. \tag{4.16}$$

В случая на прост корен (за какъвто го използваме), методът е с кубична скорост на сходимост. В случая на кратен корен сходимостта е само линейна.

4.3.2. Намиране на корените на функциите на Бесел от първи и втори вид.

Реализирана е схема за пресмятане, която използва вида на диференциалното уравнение, удовлетворявано от функциите, чиито корени се търсят. Използваната схема е с квадратична скорост на сходимост за намиране на корените на функции, които са решения на система ОДУ от първи ред, асоциирана с параметричното хомогенно линейно ОДУ от

втори ред за функциите. Този подход е изследван в [68]. Там и в [122] има подробно описание, както и доказателства за свойствата на използвания метод, а също методи, които са приложими за по-широк кръг от задачи.

[Забележка: Още един подход, използващ преобразованието на Прюфер, базиран само на вида на диференциалното уравнение, е разгледан в [71].]

Схемата за пресмятане на функции на Бесел (от първи или втори род, или тяхна линейна комбинация) е изключително проста и ще я приведем в началото, а след това ще поясним накратко, как се стига до нея от общия случай на функция, която е решение на параметрично линейно хомогенно ОДУ от втори ред. Нека примерно искаме да намерим първите N корена на функция на Бесел от първи род с индекс p, която ще означим с $J_p(x)$.

За всеки следващ корен се извършва проста итерация (fixed point iteration)

$$z_{k+1} = z_k + \arctan\left(\frac{\mathrm{J}_p(z_k)}{\mathrm{J}_{p+1}(z_k)}\right) \tag{4.17}$$

като за първоначалното приближение се използва $z_0^{(n)} = z_*^{(n-1)} + \Delta^{(n)}$, където $z_*^{(n)}$ бележи n-тия корен, а за $\Delta^{(n)}$ има два случая. Ако текущо намерените корени са по-малко от два ($n \le 1$), то $\Delta^{(n)} = \frac{\pi}{2}$, в противен случай $\Delta^{(n)} = z_*^{(n-1)} - z_*^{(n-2)}$ (това дава по-добро начално приближение).

Сега да разгледаме случая на хомогенно линейно ОДУ от втори ред

$$y'' + B_y(x)y' + A_y(x)y = 0 \tag{4.18}$$

коефициентите на което, $A_y(x)$ и $B_y(x)$, са непрекъснати в интервал *I*, чиито решения са функции *y*. Методът е приложим, когато съществуват "съпоставени" функции *w*, които са решения на уравнението

$$w'' + B_w(x)w' + A_w(x)w = 0 \tag{4.19}$$

с непрекъснати коефициенти в интервала *I* по такъв начин, че тези за всеки две двойки фундаментални решения $\left\{y^{(1)}, y^{(2)}\right\}$ и $\left\{w^{(1)}, w^{(2)}\right\}$ на двете уравнения, тези решения са свързани от системата от ОДУ от първи ред

$$\begin{cases} y' = \alpha(x)y + \delta(x)w \\ w' = \beta(x)w + \gamma(x)y \end{cases} \tag{4.20}$$

и за $\left\{y^{(1)}, w^{(1)}\right\}$ и за $\left\{y^{(2)}, w^{(2)}\right\}$ с непрекъснати коефициенти $\alpha(x)$, $\beta(x)$, $\gamma(x)$ и $\delta(x)$ в *I*.

Следва описание на преобразованията, които се извършват над тази система с цел да се получи глобално сходяща проста итерация за пресмятане на корените на $y(x)$.

Първо се извършва промяна на зависимите променливи (функциите y и w)

(4.21) $$\begin{cases} y(x) = \lambda_y(x)\bar{y}(x), \\ w(x) = \lambda_w(x)\bar{w}(x), \end{cases}$$

където $\lambda_y(x)$ и $\lambda_w(x)$ са различни от нула в I и по такъв начин, че $\bar{y}(x)$ и $\bar{w}(x)$ удовлетворяват системата

(4.22) $$\begin{cases} \bar{y}' = \bar{\alpha}\bar{y} + \bar{\delta}\bar{w} \\ \bar{w}' = \bar{\beta}'\bar{w} + \bar{\gamma}\bar{y} \end{cases}$$

с $\bar{\delta} > 0$ и $\bar{\delta} = -\bar{\gamma}$. Това се извършва с полагането

(4.23) $$\lambda_y = \operatorname{sign}(\delta)\lambda_w\sqrt{-\frac{\delta}{\gamma}}$$

и величината под квадратния корен е положителна, ако *y* и/или *w* има поне два корена в *I*. Новите функции $\bar{y}$ и $\bar{w}$ имат същите корени като *y* и *w*. Може да се провери, че $\bar{y}$ и $\bar{w}$ удовлетворяват обикновени диференциални уравнения със същите коефициенти *B* пред първата производна.

Сега след смяната на променливата

(4.24) $$z(x) = \int \bar{\delta}(x)\mathrm{d}x$$

системата придобива вида

(4.25) $$\begin{cases} \dot{\bar{y}} = \bar{a}\bar{y} + \bar{w} \\ \dot{\bar{w}} = \bar{b}\bar{w} - \bar{y}, \end{cases}$$

където $\bar{a} = \frac{\bar{\alpha}}{\bar{\delta}}$ и $\bar{b} = \frac{\bar{\beta}}{\bar{\delta}}$, а точките означават производни по отношение на z.

От последната система се вижда, че отношението $H(z) = \bar{y}/\bar{w}$ удовлетворява нелинейното ОДУ от първи ред

(4.26) $$\dot{H} = 1 + H^2 - 2\eta H\ ,$$

където $\eta = \frac{\bar{b} - \bar{a}}{2}$. Поведението на *H* прилича, поне за малки η, на поведението на функцията тангенс с редуващи се корени и сингулярности, което е следствие на коефициентите в системата (4.25).

В [122] е показано, че при дадена пропорция $H(z)$ с редуващи се корени и сингулярности, удовлетворяваща нелинейното ОДУ от първи ред (4.26), простата итерация

(4.27) $$T(z) = z - \arctan\big(H(z)\big)$$

е глобално сходяща към корените на функцията $y\big(x(z)\big)$ в интервалите, където η не променя знака си.

Сега да разгледаме случая на ОДУ от втори ред, с непрекъсната зависимост от параметър k

(4.28) $$y_k''(x)+B_k(x)y_k'(x)+A_k(x)y_k(x)=0$$

с две семейства независими решения $\left\{y_k^{(1)}\right\}$ и $\left\{y_k^{(2)}\right\}$, удовлетворяващи уравненията (смесени, диференциални с крайни разлики)

(4.29) $$\begin{cases} y_k'(x)=a_k(x)y_k(x)+d_k(x)y_{k-1}(x), \\ y_{k-1}'(x)=b_k(x)y_{k-1}(x)+e_k(x)y_k(x). \end{cases}$$

За намиране на корените на функцията y_n тези уравнения дават възможност за построяване на споменатите в началото "съпоставени" функции (ако е допустимо за параметъра k) по два начина (y_{n-1} и y_{n+1}), полагайки в уравненията (4.29) $k=n$ или $k=n+1$. Следвайки указаното по-горе, могат да се изградят две последователности прости итерации

(4.30) $$T_i(z)=z-\arctan\left(H_i(z)\right),\quad i=\pm 1,$$

където

(4.31) $$H_i(z)=-i\operatorname{sign}(d_{n_i})K_i\frac{y_n(x(z))}{y_{n+i}(x(z))},\quad \begin{cases} n, & i=-1 \\ n+1, & i=+1, \end{cases}$$

и

(4.32) $$z(x)=\int\sqrt{-d_{n_i}e_{n_i}}\,\mathrm{d}x,\quad K_i=\left(-\frac{d_{n_i}}{e_{n_i}}\right)^{\frac{i}{2}},$$

а $x(z)$ е обратната функция на $z(x)$.

Функциите $H_i(z)$ удовлетворяват $\dot{H}_i=1+H_i^2-2\eta H_i$, където $\eta_i(z)=\eta_i\left(x(z)\right)$ и

(4.33) $$\eta_i(x)=i\frac{1}{2\sqrt{-d_{n_i}e_{n_i}}}\left(a_{n_i}-b_{n_i}+\frac{1}{2}\left(\frac{e_{n_i}'}{e_{n_i}}-\frac{d_{n_i}'}{d_{n_i}}\right)\right).$$

Сега нещата могат да се конкретизират, като се сравни (4.29) с известни зависимости за функции на Бесел, например за $y_n(x)=J_n(x)$ (специален случай на по-обща зависимост за линейна комбинация на функции на Бесел от първи и втори род, например 10.6.6 в [106]).

(4.34) $$\begin{cases} J_k'(x)=-\frac{k}{x}J_k(x)+J_{k-1}(x), \\ J_{k-1}'(x)=\frac{k-1}{x}J_{k-1}(x)-J_k(x). \end{cases}$$

Ако използваме i = 1, в този случай получаваме $d_k(x) = 1$, $e_k(x) = -1$, $z(x) = x$, $K_1 = 1$ и (изпускайки индекса i)

(4.35) $$H(x) = -\frac{J_n(x)}{J_{n+1}(x)},$$

(4.36) $$T(x) = x + \arctan\left(\frac{\mathrm{J}_n(x)}{\mathrm{J}_{n+1}(x)}\right).$$

Това е всъщност простата итерация за намиране на корен на функция на Бесел, която беше формулирана в началото.

Предложеният програмен инструмент zrbessel приема от командния ред информация за тип на функцията (J или Y), индекс, брой на исканите корени и точност в десетични знаци.

4.3.3. Намиране на корените на функциите на Ейри и техните производни.

Източниците на необходимата информация тук са [106], [57].

Корените на функциите на Ейри Ai, Bi и техните производни често се срещат в асимптотични представяния на други специални функции. Всички те имат безкраен брой реални корени, които са отрицателни (отбелязване съответно с a_k, b_k, a'_k и b'_k). Поради практическата си важност, за тяхното намиране (за реалните, Bi и производната ѝ имат и комплексни) са предоставени отделни функции в основната библиотека (освен функциите за пресмятането на функциите и производните им в funcs.cs). Използва се нютонова итерация.

Изискваната втора производна при намиране на корен за производна на функция на Ейри се получава, разбира се, от вида на диференциалното уравнение на функцията: $w''(x) = xw(x)$.

Тъй като пресмятането на самите функции и производните им е относително скъпа операция, ключова е възможността за използване на много добро начално приближение, така че основната част в реализацията ни се състои в осигуряване на такова. На практика за първите корени (до номер 14) те са зададени непосредствено в кода с по 20 знака точност. За общия случай се използват равенствата ([106], 9.9(iv) и [57])

(4.37) $$\begin{cases} a_k = -T\left(\frac{3}{8}\pi(4k-1)\right), & a_k = -T\left(\frac{3}{8}\pi(4k-1)\right), \\ a'_k = -U\left(\frac{3}{8}\pi(4k-3)\right), & b'_k = -U\left(\frac{3}{8}\pi(4k-1)\right), \end{cases}$$

където $T(t)$ и $U(t)$ са съответните асимптотични представяния

(4.38) $$T(t) \sim t^{2/3}\sum_{n=0}^{\infty}\frac{T_n}{t^{2n}},\ U(t) \sim t^{2/3}\sum_{n=0}^{\infty}\frac{U_n}{t^{2n}}.$$

Първите 10 членове са съответно

n	T_n	U_n
0	1	1

1	$\frac{5}{48}$	$\frac{7}{48}$
2	$-\frac{5}{36}$	$\frac{35}{288}$
3	$\frac{77125}{82944}$	$-\frac{181223}{207360}$
4	$-\frac{108056875}{6967296}$	$\frac{18683371}{1244160}$
5	$\frac{162375596875}{334430208}$	$-\frac{91145884361}{191102976}$
6	$-\frac{1622671914671875}{66217181184}$	$\frac{91725210265629647}{3783838924800}$
7	$\frac{150126478779573265625}{82639042117632}$	$-\frac{8517284704344771067699}{4722230978150400}$
8	$-\frac{644932726927939889453125}{3470839768940544}$	$\frac{130949163695424727759631}{708334646722560}$
9	$\frac{1304211699744558907504492 1875}{520200964553048064}$	$-\frac{207878641847010708789807726484553}{8323215432848769024000}$

Таблица 4.4. Първите коефициенти в асимтотичните представяния за корените на функциите на Ейри и техните производни.

В [57] е изследвано и колко членове каква точност гарантират. Ние обаче се нуждаем само от начално приближение, което по-нататък се доуточнява итеративно.

4.4. Заключение.

Разработени са няколко програмни инструмента за пресмятане на скаларни нелинейни уравнения с много висока точност. Те обхващат реализация на бързо сходящи локални методи за случая, когато е налично достатъчно добро начално приближение, MPRootFindTestLocal, а също и реализация на глобален метод, MPRootFindTestHN. Предвид тяхната практическа важност е осъществена и самостоятелна реализация за важни частни случаи - корени на специални функции, където видът на уравнението дава допълнителна информация. Непосредствено в основния библиотечен файл funcs.cs са вградени подпрограми за частни случаи на корени на ортогонални полиноми, LaguerreZeros, HermiteZeros, LegendreZeros, както и за корени на функциите на Ейри и техните производни, Ai_zero_find, Bi_zero_find, Ai_der_zero_find, Bi_der_zero_find. Освен това са изградени и самостоятелни програмни инструменти: за намиране на корените на ортогонални полиноми, съдържащ и случая на полиноми на Якоби, zrortpol; за намиране на корените на функции на Бесел от първи и втори вид zrbessel; за намиране на корените на функциите на Ейри и техните производни, aizeros, bizeros, ai_der_zeros, bi_der_zeros.

Глава 5. Бързо пресмятане на математически константи с висока точност.

5.1. Бързо пресмятане с висока точност на някои редове. Двоично разделяне.

Ще опишем накратко използването на тази много ефективна техника, позволяваща разпаралеляване при пресмятането на някои редове от рационални числа.

Нека имаме сходящ ред с (с поне линейна скорост, n-тият член е $O(c^{-n})$ с $c > 1$) от вида

$$S = \sum_{n=0}^{\infty} \frac{a(n)}{b(n)} \frac{p(0)\cdots p(n)}{q(0)\cdots q(n)}, \tag{5.1}$$

където $a(n)$, $b(n)$, $p(n)$ и $q(n)$ са цели числа, изискващи $O(\ln(n))$ бита. На практика най-често срещан е случаят, когато $a(n)$, $b(n)$, $p(n)$ и $q(n)$ са полиноми спрямо n с целочислени коефициенти. При дадени индекси $n_1 < n_2$ разглеждаме частичната сума

$$S = \sum_{n=n_1}^{n_2-1} \frac{a(n)}{b(n)} \frac{p(0)\cdots p(n)}{q(0)\cdots q(n)}, \tag{5.2}$$

за която пресмятаме $P = p(n_1)\cdots p(n_2-1)$, $Q = q(n_1)\cdots q(n_2-1)$, $B = b(n_1)\cdots b(n_2-1)$ и $T = BQS$ по следния начин. Ако $n_2 - n_1 < 5$ (така е в оригиналната статия [74] с цел ефективност, като се ограничи малко рекурсията спрямо праволинейната реализация с $n_2 - n_1 < 3$, но тази константа подлежи на настройка), те се пресмятат директно. Ако $n_2 - n_1 \geq 5$, те се пресмятат посредством двоично разделяне (binary splitting). Избираме n_m по средата между n_1 и n_2. Пресмятаме компонентите P_l, Q_l, B_l, T_l за интервала $n_1 \leq n < n_m$, компонентите P_r, Q_r, B_r, T_r за интервала $n_m \leq n < n_2$, след което полагаме $P = P_l P_r$, $Q = Q_l Q_r$, $B = B_l B_r$ и $T = B_r Q_r T_l + B_l Q_l T_r$. Накрая прилагаме този алгоритъм за $n_1 = 0$ и $n_2 = n_{\max} = O(N)$. Очевидно трябва да се направи предварителна оценка на необходимия брой членове, гарантиращи исканата точност. Да отбележим също, че до тук всички пресмятания могат да се извършват с цели числа. Крайният резултат се получава чрез делене с плаваща запетая (само тук) $S = \frac{T}{BQ}$. Оценката за изчислителната сложност за пресмятането на S с N бита е $O\left(\left(\ln N\right)^2 M(N)\right)$, където $M(N)$ е оценката за изчислителната сложност на умножението.

Този алгоритъм е асимптотично по-бърз от праволинейното пресмятане член по член, тъй като *изтласква умноженията в област, където то е по-ефективно* (в MPIR, а и в други подобни библиотеки, има няколко прага - според броя битове на операндите - при които се сменя използваният алгоритъм за умножение). Т.е. печели се от достигнатото подобрение на $M(N)$ (ако за умножението се ползва алгоритъм с изчислителна сложност

$O(N^2)$, не би имало ускорение) и разбира се от това, че това умножение е целочислено. При наличие на няколко (ядра на) процесора, реализираният от нас метод за осъществяване на паралелни пресмятания се постига чрез разделяне на основната сума (от 0 до $n_{\max}$) на няколко частични суми, за изчислението на всяка от които се стартира отделна задача (в зависимост от броя на логическите ядра, например), след което резултатите от тях се комбинират.

Описаният алгоритъм може да се обобщи (пак [74]) за редове от вида

$$U=\sum_{n=0}^{\infty}\frac{a(n)}{b(n)}\left(\frac{c(0)}{d(0)}+\cdots+\frac{c(n)}{d(n)}\right)\frac{p(0)\cdots p(n)}{q(0)\cdots q(n)}. \tag{5.3}$$

При дадени индекси $n_1 < n_2$ разглеждаме двете частични суми

$$S=\sum_{n=n_1}^{n_2-1}\frac{a(n)}{b(n)}\frac{p(0)\cdots p(n)}{q(0)\cdots q(n)} \tag{5.4}$$

и

$$U=\sum_{n=n_1}^{n_2-1}\frac{a(n)}{b(n)}\left(\frac{c(n_1)}{d(n_1)}+\cdots+\frac{c(n)}{d(n)}\right)\frac{p(0)\cdots p(n)}{q(0)\cdots q(n)} \tag{5.5}$$

Изискванията за реда и индивидуалните коефициенти (тук има допълнително $c(n)$ и $d(n)$) са аналогични, и се пресмятат допълнително (P, Q, B и T са както по-горе) $D=d(n_1)\cdots d(n_2-1)$, $C=D\left(\frac{c(n_1)}{d(n_1)}+\cdots+\frac{c(n_2-1)}{d(n_2-1)}\right)$ и V=$DBQU$. Ако $n_2-n_1<4$, P, Q, B, T, D, C и V се пресмятат директно. Ако $n_2-n_1\geq 4$, те се пресмята чрез двоично разделяне. Избираме n_{m} по средата между n_1 и n_2. Пресмятаме компонентите P_{l}, Q_{l}, B_{l}, T_{l}, C_{l}, D_{l}, V_{l} за интервала $n_1\leq n<n_m$, компонентите P_{r}, Q_{r}, B_{r}, T_{r} C_{r}, D_{r}, V_{r} за интервала $n_m\leq n<n_2$, след което полагаме $P=P_lP_r$, $Q=Q_lQ_r$, $B=B_lB_r$, $T=B_rQ_rT_l+B_lQ_lT_r$, $C=C_lC_r$, $D=D_lD_r$ и $V=D_rB_rQ_rV_l+D_rC_lB_lP_lT_r+D_lB_lP_lV_r$. Алгоритъмът се прилага за $n_1=0$ и $n_2=n_{\max}=O(N)$, след което се извършват две деления с плаваща запетая $S=\frac{T}{BQ}$, $U=\frac{V}{DBQ}$. Изчислителната сложност за пресмятането на S и U е $O\left(\left(\ln N\right)^2 M(N)\right)$.

Някои от източниците на информация за възможностите на C# за паралелна (асинхронна) обработка са [64], [26], [65], [3]. В глава 8 е изнесен примерен код за предложената от нас конкретна реализация на паралелни пресмятания чрез двоично разделяне (за случая на константа на Апери) и списък от някои представяния на константи, подходящи за използване на двоично разделяне.

Не всички пресмятани в MPConsts константи са реализирани с паралелна обработка. Тези, които са с такава: основата на естествените логаритми e, π, константата на Ойлер-Маскерони γ, константата на Каталан, константата на Апери и ln(2). Останалите, между

които константите на Хинчин, на Рамануджан-Солднер и на Ландау-Рамануджан, се използват различни, индивидуални методи, описани по-долу. Специално за константата на Ойлер-Маскерони вж. [74] (там е отбелязана с *C*). За нея съществува и конкурентен алтернативен метод - този на Брент-МакМилан [37], който е реализиран в основната библиотека със специални функции и изчислителни методи. Пренасяне на някои от оптимизациите постигнати в MPConsts към тази основна библиотека предстои.

5.2. Една бележка за модификациите в основните библиотеки за пресмятания с произволна точност и връзките към тях.

Тъй като бързодействието тук е важна цел, наложи се да се направят някои корекции в генерирането на динамичните библиотечни файлове, съдържащи функциите на библиотеката MPIR, както и на свързващия файл xmpir.cs. В оригинала от 2010 година Sergey Bochkanov използва версии на статичните библиотеки, генерирани от доста стара версия на MPIR. Последната, 2.7.0, е от 29 юни 2015 година. Предвидена е възможност за генериране на библиотеки, които да са настроени и оптимизирани за конкретен вариант на процесора. В тази версия обаче, са промени и сигнатурите на много от функциите. Например във функциите със смесени операнди, цели с фиксирана и произволна точност, фиксираните цели числа със знак и без знак вече отговарят на C# типовете Int64 и UInt64. Това наложи многобройни промени във файловете, mpir.c, използван за генериране на динамичната библиотека и mpir.cs, използван за свързване при извикване от C# код, така че да съответстват на заглавния файл mpir.h, получен при генериране на статичната библиотека. Резултатът е вариант на динамичната библиотека, който е с около 70% процента по-бърз от оригинала за конкретния процесор.

5.3. Сравнителни резултати за няколко константи.

За сравнение ще се използва безспорния лидер към момента - програмата γ-crunher [148]. Приведени са и някои резултати от доста старата, но най-добра за времето си програма PiFast [111]. Трябва да се отбележи, че програмата γ-crunher, включваща много оптимизации (дори размера на частите, които ще се подават за паралелна обработка) е ориентирана за ефективност при изключително висока точност. Хубав обзор на използваните методики, даващ представа за цялостната картина, а и огромния обем код има в [149]. Там е и обяснението защо за по-скромни пресмятания, от порядъка на няколко хиляди или няколко стотин знака, тя не е изключителна. Тъй като една в целите на нашата система влиза използване на точност от порядък до няколко хиляди знака (за PSLQ, например това е обикновено достатъчно), макар да е способна на далеч по-широк диапазон, получените резултати, могат да се характеризират като много добри. По-нататък се подразбира, че резултатите и сравненията са за един и същ преносим компютър (TL - test laptop) със следните характеристики: Процесор: Intel(R) Core(TM) i7-3610QM CPU @ 2.30GHz (4 физически и 8 логически процесора). RAM: 16 GB. 64-битова операционна система Windows 7 Enterprise (Microsoft Windows NT 6.1.7601 Service Pack 1).

10^6 знака	e	π	ln(2)	Apéry	Lemniscate	Catalan	γ
γ-crunher	0.112	0.230	0.506	0.667	1.353	2.891	4.081
MPConsts	0.245	0.522	1.453	2.859	2.845	9.799	48.642
PiFast	0.39	1.04					

Таблица 5.1. Времената за пресмятане на неколко константи с 1 000 000 десетични знака.

10^5 знака	e	π	ln(2)	Apéry	Lemniscate	Catalan	γ
γ-crunher	0.030	0.061	0.088	0.078	0.126	0.228	0.302
MPConsts	0.039	0.050	0.134	0.207	0.186	0.612	2.510
PiFast	0.08	0.10					

Таблица 5.2. Времената за пресмятане на неколко константи с 100 000 десетични знака.

10^4 знака	e	π	ln(2)	Apéry	Lemniscate	Catalan	γ
γ-crunher	0.021	0.041	0.059	0.024	0.041	0.040	0.092
MPConsts	0.020	0.024	0.026	0.028	0.034	0.054	0.182
PiFast	0.03	0.03					

Таблица 5.3. Времената за пресмятане на неколко константи с 10 000 десетични знака.

10^3 знака	e	π	ln(2)	Apéry	Lemniscate	Catalan	γ
γ-crunher	0.021	0.033	0.050	0.033	0.039	0.025	0.086
MPConsts	0.020	0.023	0.019	0.018	0.029	0.021	0.044

Таблица 5.4. Времената за пресмятане на неколко константи с 1000 десетични знака.

500 знака	e	π	ln(2)	Apéry	Lemniscate	Catalan	γ
γ-crunher	0.021	0.034	0.052	0.034	0.033	0.024	0.067
MPConsts	0.019	0.023	0.018	0.018	0.028	0.019	0.037

Таблица 5.5. Времената за пресмятане на неколко константи с 500 десетични знака.

Границата, при която MPConsts е по-бърза е между 1 000 и 10 000 знака. За π е дори 100 000.

Данните са в секунди. Лемниската (Lemniscate) в таблиците е името на константата $\left[\Gamma\left(1/4\right)\right]^2 \Big/ \sqrt{2\pi}$. Това е единствената константа в горните таблици, която не се пресмята в MPConsts с разпаралеляване. Използва се AGM (аритметично-геометеично средно, по-конкретно изразът е $\frac{\pi}{2AGM(\sqrt{2},1)}$). PiFast не позволява пресмятане на по-малко от 10 000 знака.

5.4. Константа на Хинчин.

За дадено реално число x, чието представяне с регулярна непрекъсната дроб е

$$x = q_0 + \frac{1|}{|q_1} + \frac{1|}{|q_2} + \frac{1|}{|q_3} + \cdots, q_0 \in \mathbb{Z}, q_n \in \mathbb{N}^+, n > 0, \tag{5.6}$$

с $K(x)$ се означава границата на геометричното средно на $\{q_k\}_{k=1}^n$, т.е

$$K\left(x\right) = \lim_{n\to\infty}\left(\prod_{k=1}^{n} q_k\right)^{\frac{1}{n}}. \tag{5.7}$$

Хинчин е показал, че тази граница е еднаква (т.е. константа $K(x)=K$) за почти всички реални числа, т.е. множеството на числата за които тази граница е константа е с лебегова

мярка 1 (допълнението към това множество в $\mathbb{R}$ включва рационалните числа и квадратичните ирационални числа плюс някои други). По-конкретно

$$K = \prod_{n=1}^{\infty}\left(1+\frac{1}{n(n+2)}\right)^{\frac{\ln n}{\ln 2}}. \tag{5.8}$$

5.4.1. Един тест за NQTS.

Едно от интегралните представяния на константата на Хинчин е

$$K = 2\exp\left(\frac{1}{\ln(2)}\int_0^1 \frac{\ln\left(\Gamma(2+t)\Gamma(2-t)\right)}{t(t+1)}\mathrm{dt}\right). \tag{5.9}$$

NQTS дава за интеграла (с предварително зададена точност 60 десетични знака)

0.204271774644955698632043622429112648730122464742903249739426.

Останалата част от проверката е в SFCALC по обичайния начин. Задава му се точност 60 знака. Полученият резултат от NQTS се маркира и се копира в клипборда с Ctrl+C. В SFCALC се внася с Get. Останалите елементарни пресмятания дават за стойността на K = 2.685452001065306445309714835481795693820382293994462953051 2.

5.4.2. Резултати с MPConsts.

Пресмятането на константата на Хинчин се базира на [20]. Там са посочени конкретни резултати за пресмятане на 7350 знака за около 12 часа на работна станция IBM RS6000/ 590. Предложеният в MPConsts алгоритъм е доста праволинейна реализация на резултата от теорема 3 в [20], а именно

$$\ln(K)\ln(2) = \sum_{s=1}^{\infty}\zeta(2s,N)\frac{A_s}{s} - \sum_{k=2}^{\infty}\ln\left(1-\frac{1}{k}\right)\ln\left(1+\frac{1}{k}\right), \tag{5.10}$$

където $\zeta(s,N) = \sum_{n=1}^{\infty}\frac{1}{(n+N)^s}$ е функцията на Хурвиц, която за неотрицателно цяло число N е $\zeta(s) - \sum_{n=1}^{N}\frac{1}{n^s}$, а $A_s = \sum_{m=1}^{2s-1}\frac{(-1)^{m-1}}{m}$.

За пресмятане на ς функцията с четен целочислен аргумент се използва формулата

$$\zeta(2n) = (-1)^{n-1}\frac{2^{2n}B_{2n}}{2(2n)!}\pi^{2n} = \frac{T_n}{2^{2n+1}(1-2^{-2n})(2n-1)!}\pi^{2n}, \tag{5.11}$$

където B_{2n} са числата на Бернули, а T_n са тангенсовите числа. Връзката между тях е

$$T_n = \frac{(-1)^{n-1}2^{2n}(2^{2n}-1)}{2n}B_{2n}, \tag{5.12}$$

а алгоритъмът е TangentNumbers [36]:

Input: positive integer m
Output: Tangent numbers $T_1,\ldots,T_m$
  $T_1 \leftarrow 1$
  for k **from** 2 **to** m **do**
    $T_k \leftarrow (k-1)T_{k-1}$
  for k **from** 2 **to** m **do**
    for j **from** k **to** m **do**
      $T_j \leftarrow (j-k)T_{j-1} + (j-k+2)T_j$
  return $T_1,\ldots,T_m$

Фигура 5.1. Алгоритъмът TangentNumbers.

В [1] е приведен резултат с 10025 знака получен в Python (mpmath+gmpy) за 11 минути (използваната машина и версиите на софтуера не са посочени). MPConsts получава този резултат на TL за около 33,5 секунди. На адрес [73] има резултат от 1998 година (Xavier Gourdon) за 110000 знака. Този резултат е записан като рекорден (в [104] последно обновено на 12 август 2010 година). Машината е sgi R10000. Времето за изчисление е 22 часа и 23 минути. MPConsts получава K с тази точност за 5 часа и 8 минути. Поставянето на рекорди не е самоцел, така че решихме да не губим време с пресмятане на повече знаци.

При реализацията в MPConsts необходимите числа на Бернули (вероятно завишен) се определя емпирично. Очевидно са възможни оптимизации, предвид факта, че времето за изпълнение (за 110000 знака) е само около 4,4 пъти по-малко (отношението в бързината на използвания процесор е 9,2).

Използваният алгоритъм за пресмятане на числата на Бернули е квадратичен. За числата на Бернули съществуват и по-бързи алгоритми, в случай че става дума за едно число (вж. например [78]). В дадения случай обаче, се пресмятат всички до определен граница.

5.5. Константа на Раманудджан-Солднер.

Определя се като единствения положителен корен на интегралния логаритъм $\mathrm{li}(x) = \int_0^x \frac{\mathrm{dt}}{\ln(t)}$. За намирането му се използва част от част от друг програмен инструмент, а именно MPRootFindTestLocal, който е описан в главата за намиране на корени на уравнения. Непосредствено в кода е зададено добро начално приближение, ускоряващо решението.

1000 десетични знаци	10 000 десетични знаци	100 000 десетични знаци
0.080 секунди	0.949 секунди	268.564 секунди

Таблица 5.6. Времената за пресмятане на константата на Раманудджан-Солднер.

5.6. Константа на Ландау-Рамануджан.

Нека с $B(x)$ означим броя на положителните цели числа по малки от x, които се представят като сума от два точни квадрата. В теорията на числата има резултат, според който $B(x)=O\left(\frac{x}{\sqrt{\ln(x)}}\right)$. Границата $\lambda=\lim_{n\to\infty}\frac{B(x)\sqrt{\ln(x)}}{x}$ се нарича константа на Ландау-Рамануджан. Ефективно пресмятане на тази константа е възможно, само ако съществува някакво аналитично представяне. Такова има:

$$\lambda=\frac{1}{\sqrt{2}}\prod_{n=1}^{\infty}\left[\left(1-\frac{1}{2^{2^n}}\right)\frac{\zeta\left(2^n\right)}{\beta\left(2^n\right)}\right]^{\frac{1}{2^{n+1}}}, \tag{5.13}$$

открито от Шенкс [123]. В израза освен ς-функцията на Риман присъства β-функцията на Дирихле $\beta(s)=\sum_{k=0}^{\infty}\frac{(-1)^k}{(2k+1)^s}$. Едно елегантно извеждане има в [63]. Предложената в MPConsts реализация е с идеята изчисляваните стойности да зависят по възможност от стойностите на ς-функцията на Риман за четни положителни числа, за които има представяне чрез числата на Бернули (вж. по-горе в подраздела за константата на Хинчин). За целта се използват равенствата ([89], [130])

$$\begin{aligned}\beta(2n)&=(-1)^{n+1}\frac{(\pi/2)^{2n-1}}{(2n-1)!}\ln 2\\&-(-1)^{n+1}\sum_{k=1}^{n-1}(-1)^k\frac{(\pi/2)^{2n-2k-1}}{(2n-2k-1)!}\left(1-\frac{1}{2^{2k}}\right)\zeta(2k+1)\\&+(-1)^n\frac{\pi^{2n-1}}{2^{2n}}\sum_{k=0}^{\infty}\frac{\zeta(2k+2)}{2k(2k+1)\ldots(2k+2n-1)}\left(\frac{1}{2^{2k}}-\frac{1}{2^{4k}}\right)\end{aligned} \tag{5.14}$$

и

$$\zeta(2n+1)=(-1)^n\frac{2(2\pi)^{2n}}{(2n-1)2^{2n}+1}\left[\sum_{k=1}^{n-1}\frac{(-1)^{k-1}k}{(2n-2k+1)!}\frac{\zeta(2k+1)}{\pi^{2k}}+\sum_{k=0}^{\infty}\frac{(2k)!}{(2n+2k+1)!}\frac{\zeta(2k)}{2^{2k}}\right] \tag{5.15}$$

Може да се използва факта ([63]), че $\left(1-2^{-s}\right)\frac{\zeta(s)}{\beta(s)}=\prod_r\frac{1+r^{-s}}{1-r^{-s}}$, където r пробягва стойностите на простите числа, чийто остатък при деление на 4 е 3. Т.е.

$$\lambda\sqrt{2}=\prod_{n=1}^{\infty}\left[\prod_r\frac{1+r^{-2^n}}{1-r^{-2^n}}\right]^{\frac{1}{2^{n+1}}}, \tag{5.16}$$

$$\ln(\lambda\sqrt{2}) = \sum_{n=1}^{\infty}\frac{1}{2^{n+1}}\ln\prod_{r}\frac{1+r^{-2^n}}{1-r^{-2^n}} = \sum_{n=1}^{\infty}\frac{1}{2^{n+1}}\sum_{r}\ln\left(\frac{1+r^{-2^n}}{1-r^{-2^n}}\right) =$$

(5.17)

$$= \sum_{n=1}^{\infty}\sum_{r}\frac{1}{2^{n+1}}\ln\left(\frac{1+r^{-2^n}}{1-r^{-2^n}}\right) = \sum_{r}\sum_{n=1}^{\infty}\frac{1}{2^{n+1}}\ln\left(\frac{1+r^{-2^n}}{1-r^{-2^n}}\right)$$

На даден етап (голяма стойност на r^{2^n}), може да се пресмята непосредствено дадена вътрешна сума.

Реализацията в MPConsts не е съвършена и подлежи на оптимизации или дори смяна на подхода, но работи коректно. 4000 знака се постигат за 44 минути и 38 секунди, спрямо 5 минути и 16 секунди за 2000 знака и 37 секунди за 1000 знака. Като сравнение за малък брой десетични знаци, в [63] се посочва, че 200 знака са пресметнати за по-малко от 6 секунди (1996 година). С MPConsts времето е около 0.7 секунди.

5.7. Граница на Лаплас.

Задачата водеща до тази константа произлиза от небесната механика. Уравнението на Кеплер $M = E - \varepsilon\sin E$ свързва средната аномалия M с ексцентричната аномалия E, за тяло, което се движи по елипса с ексцентричност ε. Уравнението не се решава с елементарни функции, но теоремата за обръщане на Лагранж дава решение като ред по степените на ε. Лаплас е открил, че този ред е сходящ за стойности на ε по-малки от дадена величина и е разходящ за по-големи. Тази величина (радиусът на сходимост на съответния ред) се нарича граница на Лаплас. За целите на нейното изчисляване е важно да се знае, че тя е решение на уравнението $\dfrac{xe^{\sqrt{1+x^2}}}{1+\sqrt{1+x^2}} - 1 = 0$.

В MPConsts директно е кодирана нютонова итеративна стъпка за това уравнение, както и зададено в кода добро начално приближение (както при константата на Рамануджан-Солднер). Резултати съответно: 10 000 знака се постигат за 0.175 секунди, 100 000 за 15.2 секунди , а 1 000 000 за 12 минути и 32 секунди.

5.8. Заключение.

Разработени са методи за бързо пресмятане на някои математически константи с висока точност. Предложените методи за реализация са разнообразни, в зависимост от пресмятаната константа. В някои случаи е възможно прилагането на изключително ефективното двоично разделяне (binary splitting), в други случаи се използват методи, специално адаптирани за пресмятане на конкретна константа.

Глава 6. Метод за определяне на целочислена зависимост и идентификация на константи. Алгоритъм PSLQ.

6.1. Описание на стандартния (базов) вариант на алгоритъма PSLQ.

PSLQ построява поредица от матрици с целочислени елементи B_n които формират "редуцирания" вектор $y = xB_n$, или докато се намери целочислена зависимост (като една от колоните на B_n), или докато се изчерпи наличната точност. Освен първоначалния n-мерен вектор x, съдържащ числата, за които се търси целочислена зависимост, в алгоритъма се използва още един вектор y със същата размерност, $n \times n$ матрици A, B и $n \times (n-1)$ матрица H. Избира се също параметър $\gamma \geq \sqrt{\frac{4}{3}}$ (в реализацията $\gamma = \sqrt{\frac{4}{3}}$). nint е функцията, даваща най-близкото цяло число.

// Начало на алгоритъма

Инициализация:

Стъпка 1. // $A = B = I_{(n)}$
for $j := 1$ to n:
  for $i := 1$ to n:
    if $i = j$ then $A_{ij} := 1$; $B_{ij} := 1$;
    else $A_{ij} := 0$; $B_{ij} := 0$;
    endif;
  endfor;
endfor;

Стъпка 2.
for $k := 1$ to n:
  $s_k := \sqrt{\sum_{j=k}^{n} x_j^{\,2}}$;
endfor;
$t := 1/s_1$;
for $k := 1$ to n:
  $y_k := tx_k$;
  $s_k := ts_k$;
endfor;

Стъпка 3. // Инициализация на H
for $j := 1$ to n - 1:
  for $i := 1$ to j - 1:
    $H_{ij} := 0$;
  endfor;

for $i := j+1$ to *n*:

$H_{ij} := -y_i y_j \big/ \left(s_j s_{j+1}\right)$;

endfor;

endfor;

Стъпка 4. // Редуциране на *H*

for $i := 2$ to *n*:

for $j := i-1$ to 1 step -1:

$t := \text{nint}\left(H_{ij} \big/ H_{jj}\right)$;

$y_j := y_j + t y_i$;

for $k := 1$ to *j*:

$H_{ik} := H_{ik} - t H_{jk}$;

endfor;

for $k := 1$ to *n*:

$A_{ik} := A_{ik} - t A_{jk}$;

$B_{kj} := B_{kj} + t B_{ki}$

endfor;

endfor;

endfor;

Итерация:

Стъпка 1.

Избира се такова *m*, за което $\gamma^i \left|H_{ii}\right|$ е максимално при *i* = *m*.

Стъпка 2.

Във вектора *y* се разменят местата на елементите с индекси *m* и *m* + 1, както и съответните редове на *A* и *H* и колони на *B* за същите индекси.

Стъпка 3. // Премахване на ъгъла за диагонала на *H*:

If $m \le n-2$ then

$t_0 := \sqrt{H_{mm}^2 + H_{m,m+1}^2}$;

$t_1 := H_{mm} \big/ t_0$;

$t_2 := H_{m,m+1} \big/ t_0$;

for $i := m$ to *n*:

$t_3 := H_{im}$;

$t_4 := H_{i,m+1}$;

$H_{im} := t_1 t_3 + t_2 t_4$;

$H_{i,m+1} := -t_2 t_3 + t_1 t_4$;

endfor;

endif;

Стъпка 4. // Редуциране на *H*:
for $i := m+1$ to *n*:
  for $j := \min(i-1, m+1)$ to 1 step -1:
    $t := \text{nint}\left(H_{ij} / H_{jj}\right)$;
    $y_j := y_j + t y_i$;
    for $k := 1$ to *j*:
      $H_{ik} := H_{ik} - t H_{jk}$;
    endfor;
    for $k := 1$ to *n*:
      $A_{ik} := A_{ik} - t A_{jk}$;
      $B_{kj} := B_{kj} + t B_{ki}$;
    endfor;
  endfor;
endfor;

Стъпка 5. // Граница за нормата
$M := 1 / \max_j \left| H_{jj} \right|$.
Не може да има вектор на съотношението, чиято евклидова норма е по-малка от *M*.

Стъпка 6. // Тест за завършване
Ако най-големият елемент на *A* надхвърля нивото на използваната точност, точността е изчерпана. Ако най-малкият елемент на вектора *y* е по-малък от прага за откриване (detection threshold, вж. по-долу в забележките как се задава), намерено е съотношение, което е дадено в съответната колона на *B*.

// Край на алгоритъма

Фигура 6.1. Алгоритъмът PSLQ.

Въпреки ефективността на PSLQ (изчислителна сложност от порядъка на $O(n^3)$), все пак времето за пресмятане силно нараства с нивото на прилаганата точност на пресмятанията. По принцип, за да се открие целочислена зависимост на *n* числа, зададени с точност от *d* десетични знака, за точността на елементите на входния вектор *x* трябва да се зададе поне *nd*, а при практическата реализация малко повече: 10 до 15%. При откриване на зависимост, най-малкият елемент на вектора *y* рязко намалява до стойност от порядъка на съответния "епсилон за изчисленията" (т.е около 10^{-p}, където *p* е точността, с която се извършват пресмятанията). "Прагът на откриване" в стъпка 6 на итерацията по-горе обикновено се задава да е само с няколко порядъка по-голям от съответния епсилон на пресмятанията, позволявайки да се компенсира евентуална грешка, натрупвана от закръглявания. Отношението между най-големия и най малкия елемент във вектора *y* при откриване на целочислено съотношение може да си приеме за "ниво на достоверност".

6.2. Практическа реализация.

PSLQ съществува в два основни варианта - базов и така наречения "multi-pair" ([23], раздел 2). В предлаганата програмна система е реализиран базовият вариант. PSLQ реализацията за момента е на основно равнище, т.е. работи с вход, който е набор от константи, зададени от потребителя (във вид на файл, в чийто първи ред е записан броят на използваните константи, вторият е използваната точност, а следващите редове са самите константи в текстов вид).

Както бе споменато по-горе, алгоритъмът извършва пресмятанията с точност от порядъка на nd (d е точността на представяне на числата, n е броят им). Така че стратегията за нанасяне във входния набор за търсене освен на изследваната и на всевъзможни други константи (плюс евентуално техни мултипликативни комбинации) не е работещ вариант. Изследващият трябва да се ограничи до това, което му подсказва интуицията, и/или да предвиди някакви евристики, отсяващи достъпните за пораждане константи, така че входния набор да е с приемлив размер (точността на пресмятането да е ограничена до приемлива граница за използваната компютърната система). Освен това, за самите константи трябва да съществуват програмни средства, която да я поражда ефективно с достатъчно висока точност. Това беше и един от мотивите за създаването на програмния инструмент NQTS за пресмятане на определени интеграли, както и реализацията за бързо пресмятане на константи, чийто списък очевидно би трябвало да се разширява.

Възможната автоматизация, в която входните константи да се пораждат при посочване автоматично в потребителски интерфейс, както и евентуални допълнителни евристики, предстои евентуално да бъдат правени в бъдеще, така че да се реализира програмен инструмент от типа identify(x), за константа x, достъпно в някои системи с елементи на компютърна алгебра. Макар и да изискват известна изобретателност, идеите, използвани от такива програмни инструменти са известни [28] и могат да бъдат сравнително лесно реализирани. Колкото до усъвършенстван вариант на PSLQ ("multi-pair", например), въпросът стои по малко по-различен начин. Алгоритъмът е доста труден за модификация с паралелни пресмятания и макар такава да е правена [22], тя е специфична. Разпаралеляване в конкретната среда на предлаганата система би трябвало да се извърши и със специфичните средства предлагани от тази среда.

Конкретен прост пример за приложение в системата е даден в раздела за NQTS. Примерът, разбира се, е донякъде изкуствен. В предварителния набор освен резултата от NQTS за

$$\int_0^1 \frac{t^2 \ln t}{(t^2-1)(t^4+1)} \mathrm{d}t \left(= \frac{\pi^2(2-\sqrt{2})}{32} \right). \tag{6.1}$$

трябва да присъстват π^2 и $\pi^2\sqrt{2}$. Пресмятанията са с точност 100 десетични знака. Резултат се получава за 9 итерации.

По-реалистичен пример, тестващ работоспособността на реализирания алгоритъм, е споменатият в уводната глава пример с намиране на полинома от 12-та степен, чийто корен е

$r = B_3 =$
3.544090359551922853615965986604804540583099845444573675457812530305842942858863012256258566424891799962 6... (третата бифуркация на логистичното изображение).

При точност на пресмятанията 1570 десетични знака (1.15*13*105), решението се получава за 1308 итерации (4-5 секунди, включващи и извеждането на тестова информация от по няколко реда за всяка итерация).

Както бе споменато по-горе, едно от изискванията при идентификация на константи е възможността за генериране на самата константа. В случая не разполагаме с програма за генериране на величините B_n, при стойностите на които за параметъра r възниква бифуркация на логистичното изображение. Приведената по-горе числена стойност за B_3 е взета от [14].

6.3. Заключение.

Разработен е метод за ефективна програмна реализация на алгоритъма PSLQ в неговия основен вариант за настоящата програмна среда. Работоспособността е проверена за нетривиални примери.

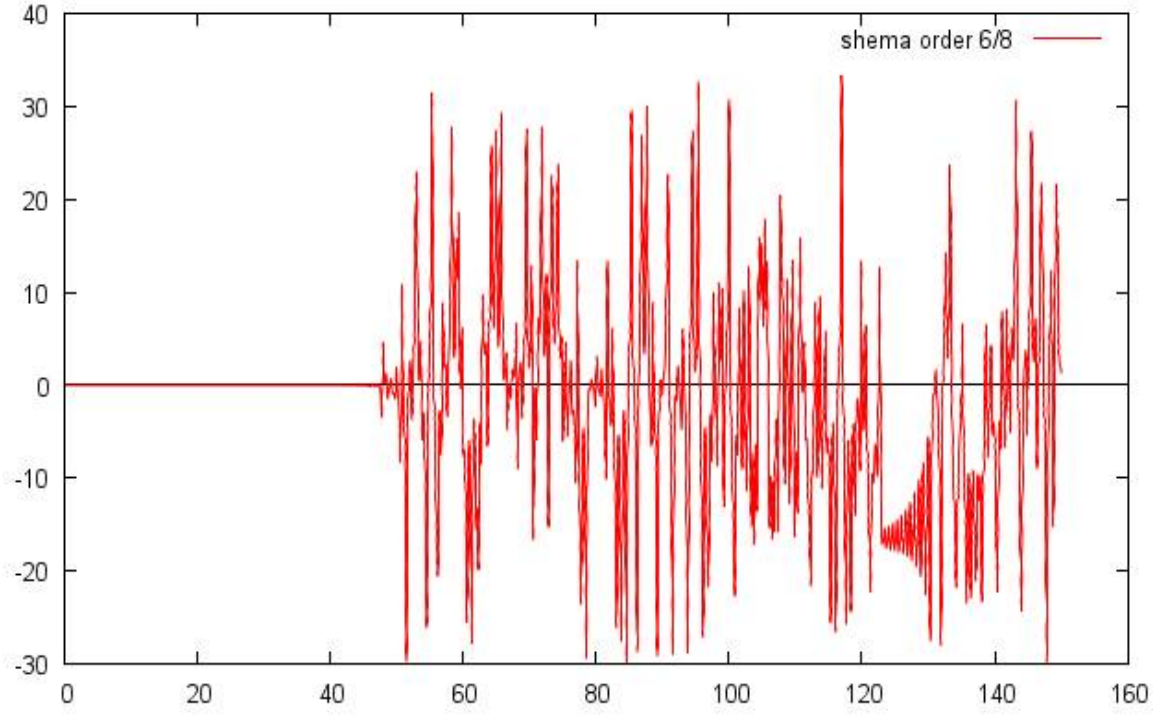
Глава 7. Приложения свързани с пресмятането на системи ОДУ с висока точност.

7.1. Пример за пресмятане с все по-голяма точност на динамична система с хаотично поведение.

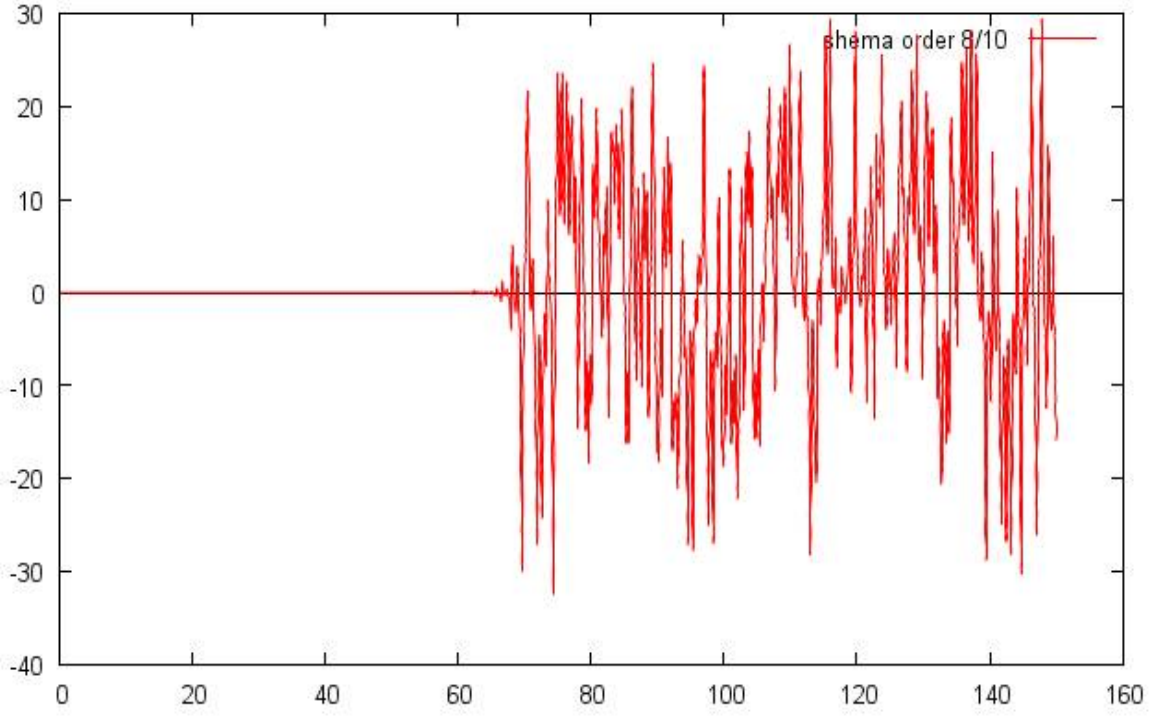
Като илюстрация към раздела 3.1, ето как стоят нещата на практика (дали е безнадеждно е въпрос на гледна точка) при пресмятане на знаменитата система на Лоренц

$$\begin{aligned} \dot{x} &= \sigma y - \sigma x \\ \dot{y} &= rx - y - xz \\ \dot{z} &= -bz + xy \end{aligned} \qquad (7.1)$$

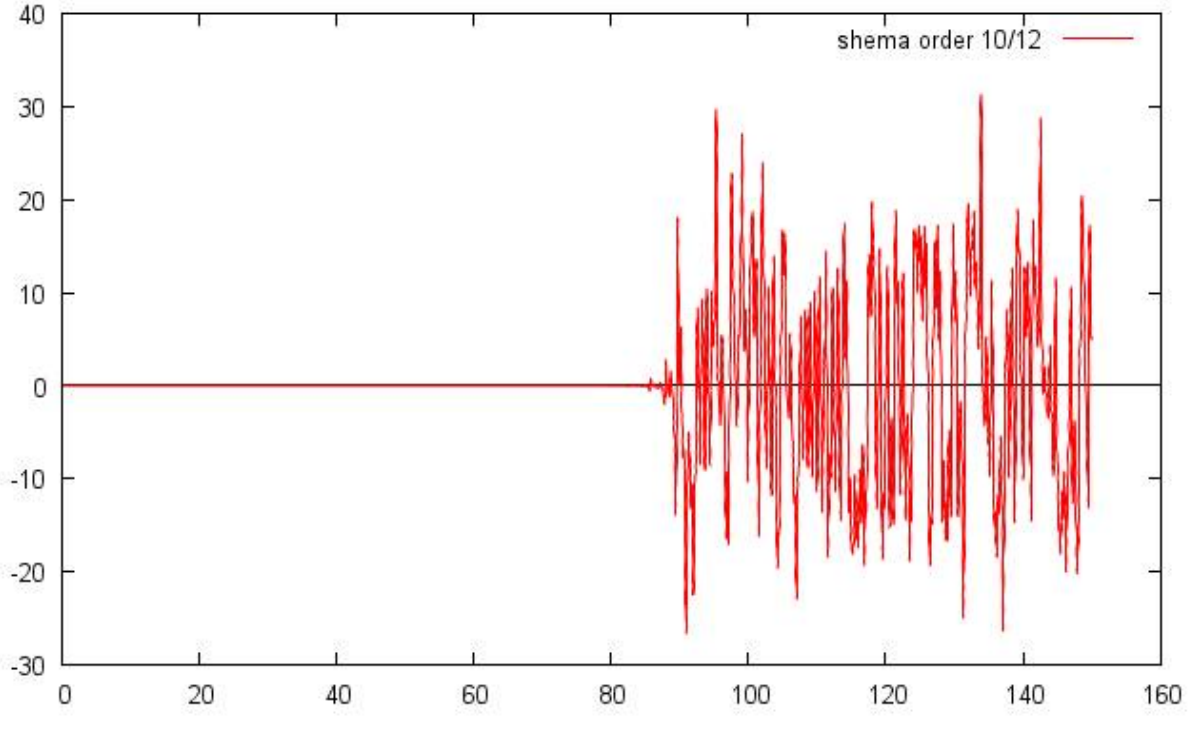
със стандартен избор на коефициентите $\sigma = 10, b = 8/3, r = 28$. За начална точка е избрана (10.6451, 4.06125, 36.057) – в близост до атрактора, за да не се пресмята преходният период.



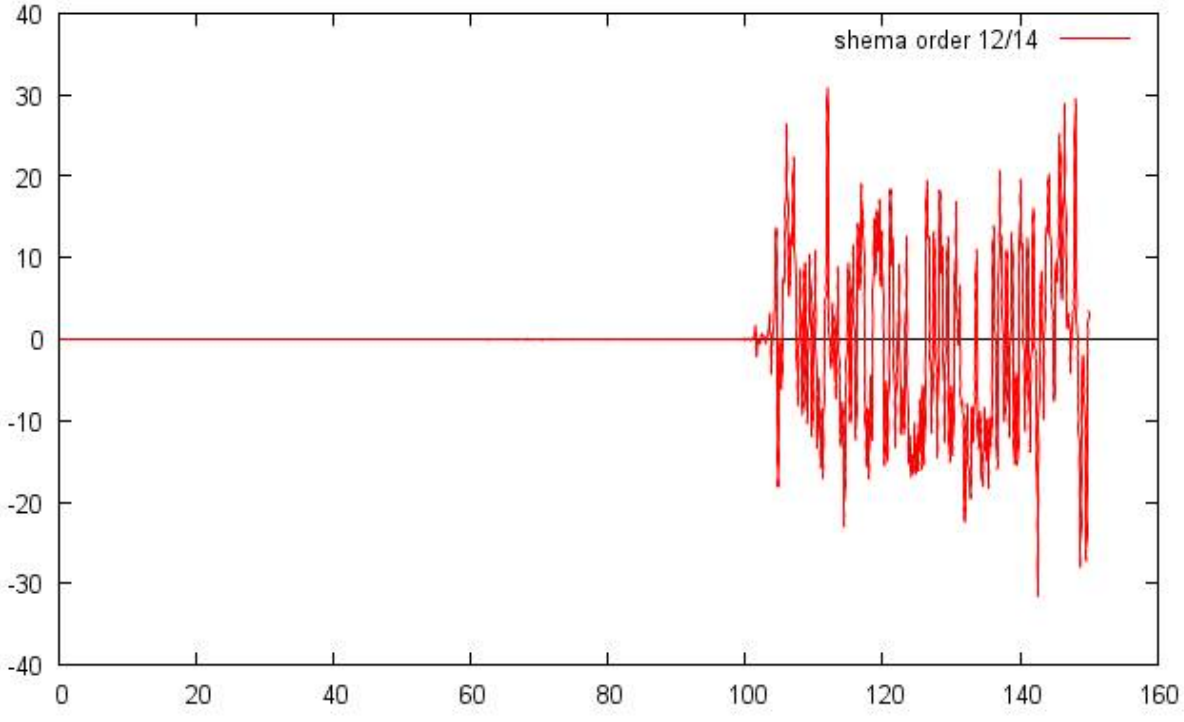
Фигура 7.1. Разлика на *x*-компонентата на решението на системата на Лоренц при пресмятане с неявна схема на Рунге-Кута от ред 6 и 8 със стъпка 1/100 при точност 120 десетични знака за 15 000 стъпки.



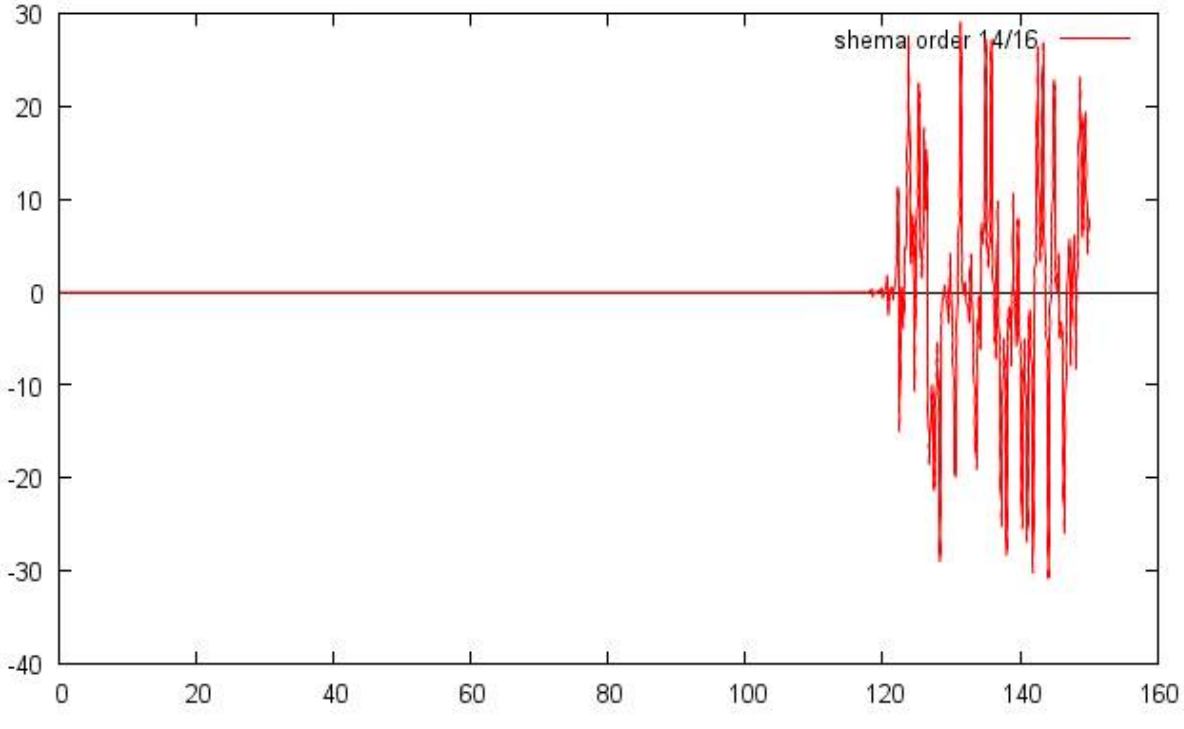
Фигура 7.2. Разлика на *x*-компонентата на решението на системата на Лоренц при пресмятане с неявна схема на Рунге-Кута от ред 8 и 10 със стъпка 1/100 при точност 120 десетични знака за 15 000 стъпки.



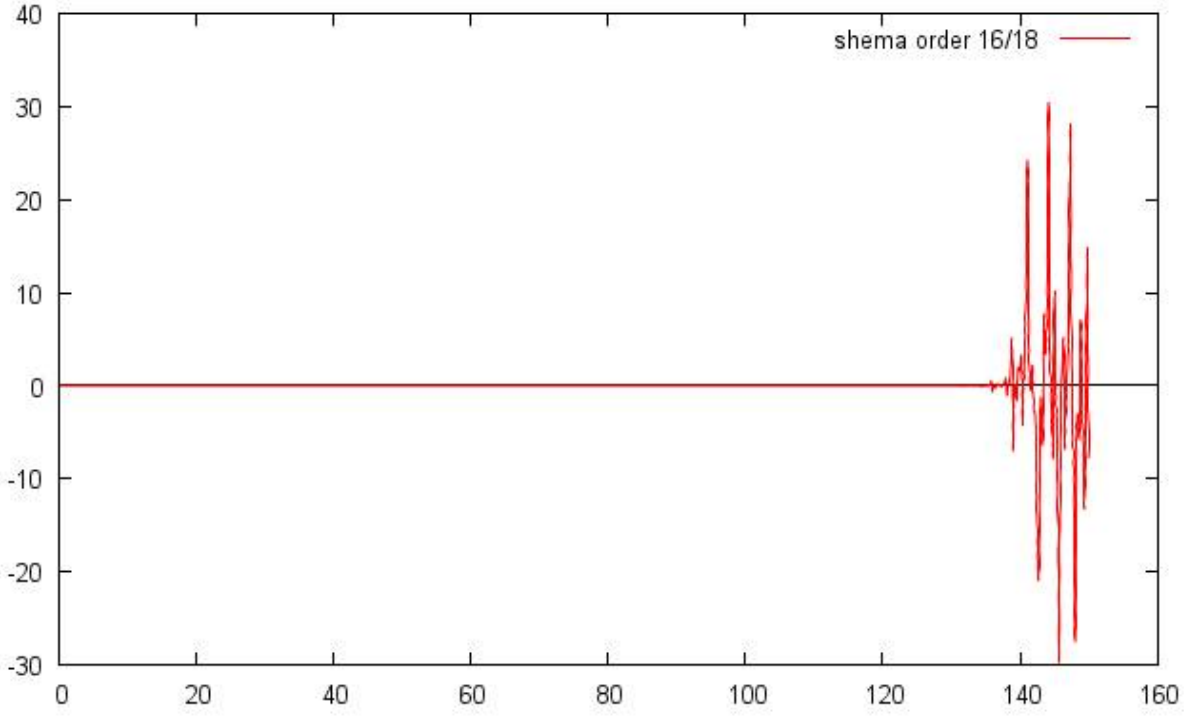
Фигура 7.3. Разлика на *x*-компонентата на решението на системата на Лоренц при пресмятане с неявна схема на Рунге-Кута от ред 10 и 12 със стъпка 1/100 при точност 120 десетични знака за 15 000 стъпки.



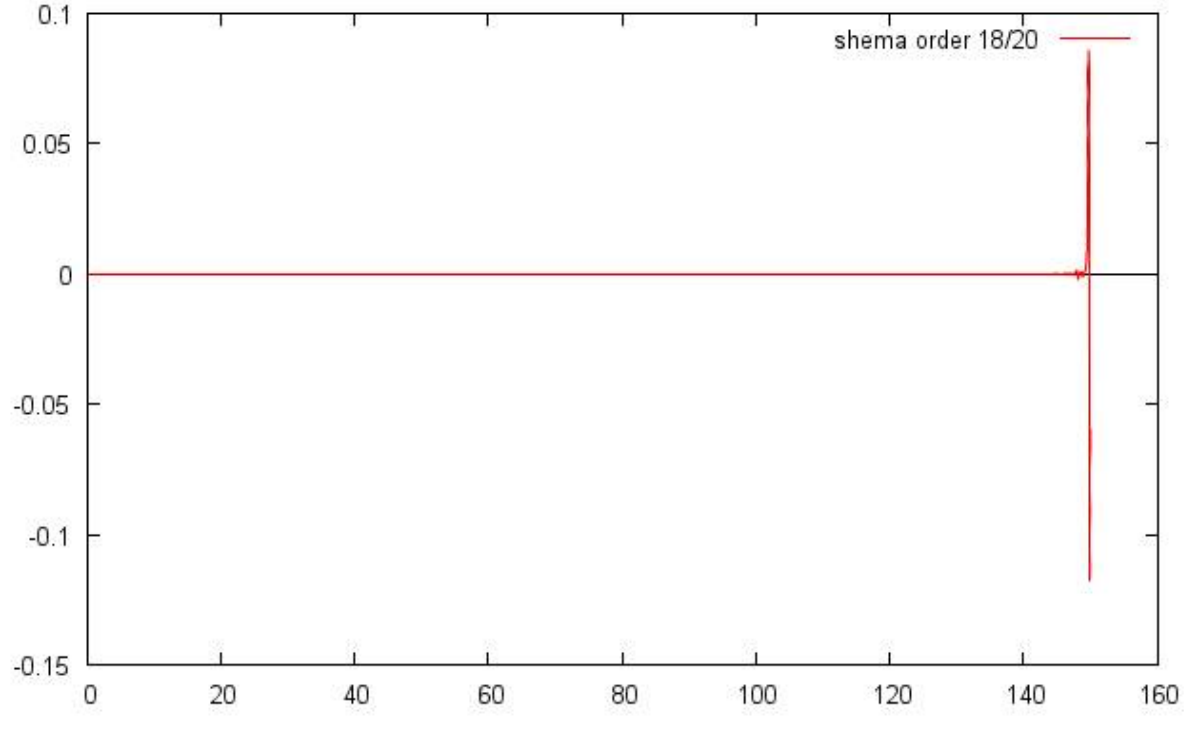
Фигура 7.4. Разлика на *x*-компонентата на решението на системата на Лоренц при пресмятане с неявна схема на Рунге-Кута от ред 12 и 14 със стъпка 1/100 при точност 120 десетични знака за 15 000 стъпки.



Фигура 7.5. Разлика на *x*-компонентата на решението на системата на Лоренц при пресмятане с неявна схема на Рунге-Кута от ред 14 и 16 със стъпка 1/100 при точност 120 десетични знака за 15 000 стъпки.



Фигура 7.6. Разлика на *x*-компонентата на решението на системата на Лоренц при пресмятане с неявна схема на Рунге-Кута от ред 16 и 18 със стъпка 1/100 при точност 120 десетични знака за 15 000 стъпки.



Фигура 7.7. Разлика на *x*-компонентата на решението на системата на Лоренц при пресмятане с неявна схема на Рунге-Кута от ред 18 и 20 със стъпка 1/100 при точност 120 десетични знака за 15 000 стъпки.

Графики показват разликата на *x* (за *y* и *z* картината е сходна) при пресмятане с предвидения в системата решател на обикновени диференциални уравнения, реализиращ неявна (с порядък по избор) схема на Рунге-Кута при различен порядък на схемата (показван горе вдясно на всяка графика). Всички пресмятания са правени със стъпка 0.01 и с 120 десетични знака точност за 15 000 стъпки (интервал от 150 условни единици време).

Естествено, за по-нататъшно екстраполиране на интервала на "предвидимост" в даден момент освен порядъкът на схемата, трябва да се увеличи и точността на пресмятанията (а вероятно и да се намали стъпката). Не са приведени картинки в равнините xy, xz, yz, tx, ty, tz - изглеждат сходно за различна точност на пресмятанията.

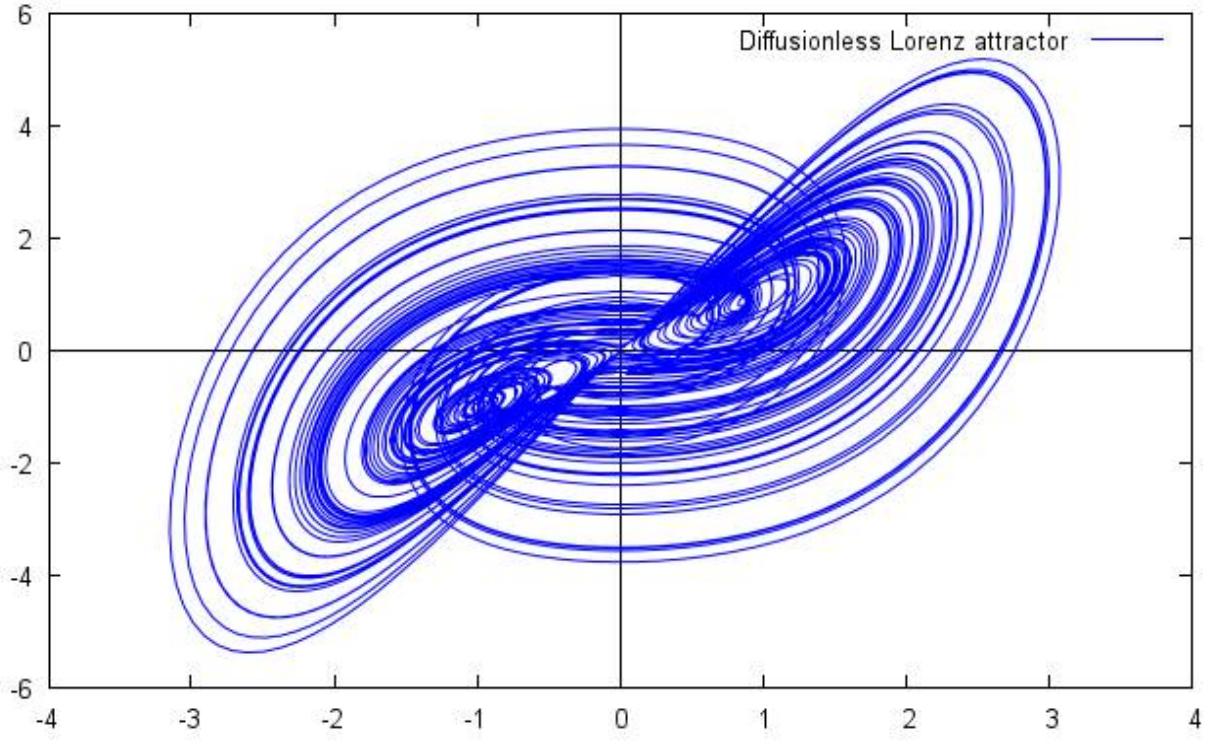
7.2. Илюстрации от числено пресмятане на някои много прости гладки динамични системи с хаотично поведение.

Използвали сме проста схема, при която системата се пресмята с основния за системата решател на системи обикновени диференциални уравнения, а за визуализация се използва програмата gnuplot (от изпълним файл, подобно на описаното при разглеждане на програмата VODE2, но без динамично визуализиране по време на решаване, както е във VODE2 – просто се използва файлът със записаното решение, в случая „irk_test.dat" или „irk_test_d.dat"). Качествено получените илюстрации не зависят твърде много от използваната точност и порядък на неявната схема Рунге-Кута, така че като допълнение към основната библиотека бе добавен вариант на решателя, който работи с double. Това – за случаите, в които трябва да се „проиграят" голямо количество различни примери. За по-прецизни цели, като например изследване на точността на пресметнатото решение, подобно на това в предишното приложение към тази глава, "основният решател" (с произволно задавана точност на пресмятанията) е необходим. По-долу са приведени примерни системи уравнения и съответните им решения, проектирани в равнина xy. Примерите в първата част на това приложение са взети от третата глава (Autonomous Dissipative Systems) на [165]. Всъщност, там с изключение на главите 8 и 9 ("Spatiotemporal Systems" и "Time-Delay Systems"), които изискват други средства за решаване, на практика всички останали примери могат да служат за тестове, подобно на приведените. Почти всички решения са за 10 000 стъпки (с няколко изключения, където стъпката е по-малка а броят на стъпките е по-голям), големина на стъпката 1/25 и порядък 4 на неявната схема на Рунге-Кута. За почти всички илюстрации е използван „основният решател" (картинките бяха записани преди добавянето на опростения). Използвана е точност 30 десетични знака. Навсякъде в илюстрациите участват компонентите *x* и *y* или съответно x и $\dot{x}$, ако се решава система, получена от едно уравнение.

Атрактор на Лоренц „без дифузия". Класическата система на Лоренц не може да се опрости допълнително (полагайки нула някой от членовете в дясната страна на уравненията), без да се загуби хаотичното поведение. Обаче премащабирането $(x,y,z)\to(\sigma x,\sigma y,\sigma z+r),\ t\to t/\sigma$ с последващо устремяване на *r* и σ към безкрайност, но по такъв начин, че $R=br/\sigma^2$ да остава крайно, дава така наречената бездифузна система на Лоренц

$$\begin{cases} \dot{x} = y - x, \\ \dot{y} = -xz, \\ \dot{z} = xy - R, \end{cases} \tag{7.2}$$

която е хаотична за широк диапазон от стойности на единствения параметър R, включително за стойност единица, за която е приведената илюстрация (с начални стойности $(x_0, y_0, z_0) = (1,0,1)$).

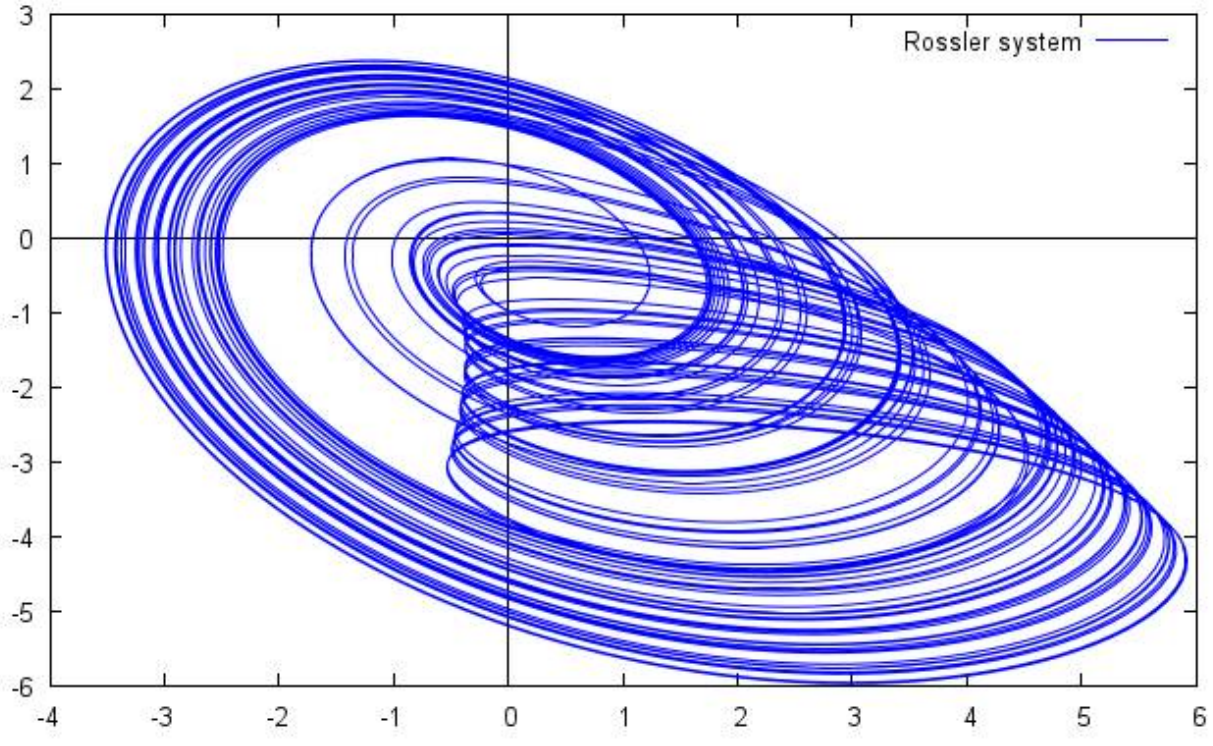


Фигура 7.8. Система на Лоренц "без дифузия".

Системата на Рьослер (Rössler). Бездифузната система на Лоренц има все пак два нелинейни члена. Системата на Рьослер е само с един такъв,

$$\begin{cases} \dot{x} = -y - z, \\ \dot{y} = x + ay, \\ \dot{z} = b + z(x - c). \end{cases} \tag{7.3}$$

За илюстрацията са използвани стойности на параметрите $(a,b,c) = (0.5,1,3)$, за които с-мата е хаотична, а началните стойности са $(x_0, y_0, z_0) = (2,0,1)$.



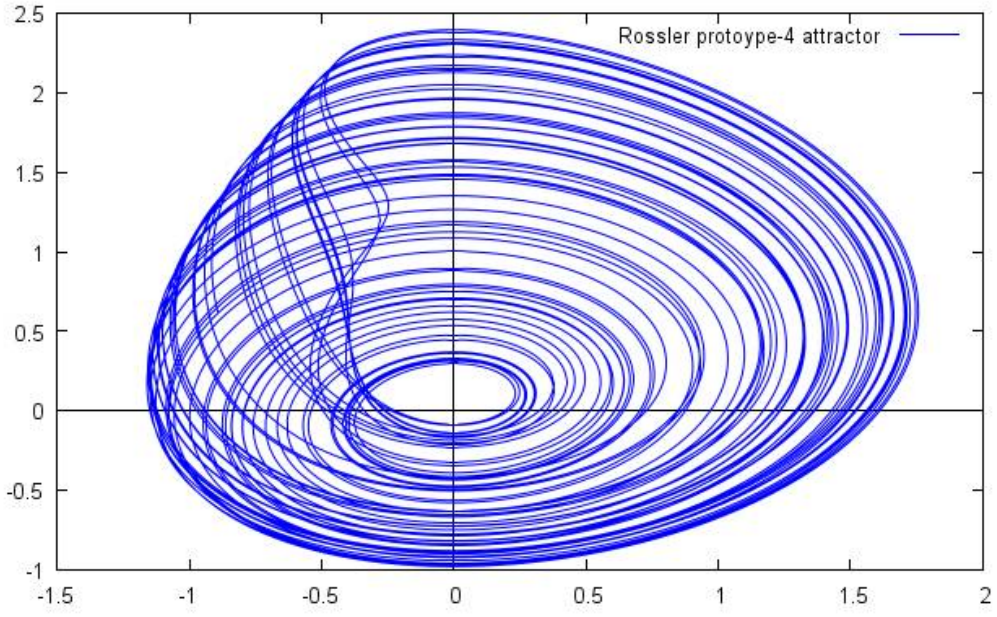
Фигура 7.9. Система на Рьослер.

Система на Рьослер: прототип-4

$$\begin{cases} \dot{x} = -y - z, \\ \dot{y} = x, \\ \dot{z} = a(y - y^2) - bz. \end{cases} \tag{7.4}$$

Тази система има само 6 члена с два параметъра и една квадратична нелинейност (y^2). Приведената илюстрация е за $(a,b) = (0.5, 0.5)$, за която с-мата е с хаотично поведение, и с начални стойности $(x_0, y_0, z_0) = (0.1, 0.3, 0)$.

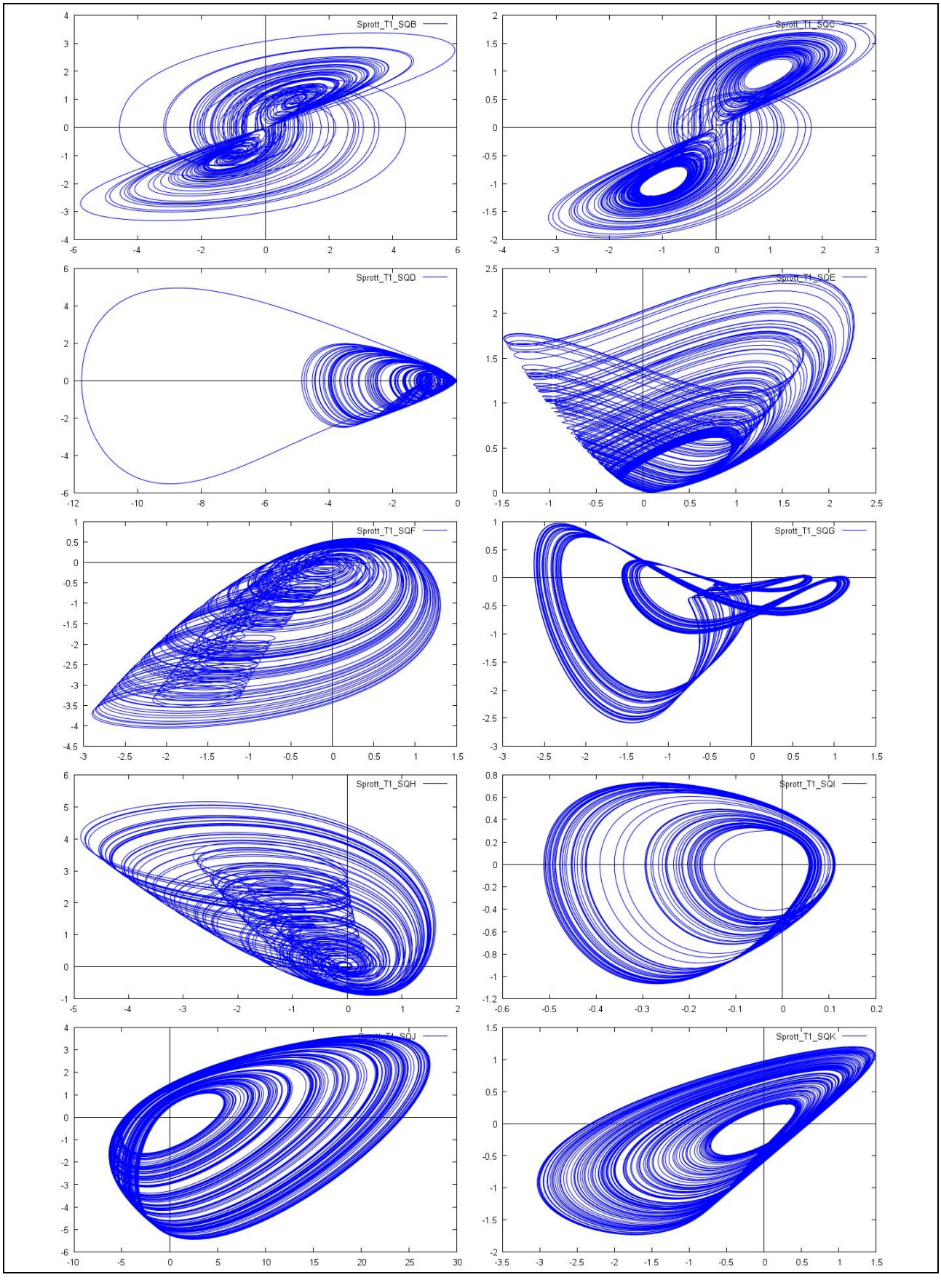
Следват илюстрации на 16 прости тримерни системи с квадратични нелинейности (следват отляво надясно и отгоре надолу системите в таблицата, такъв е редът на представяне и по-нататък). Началните условия са показани в таблицата.

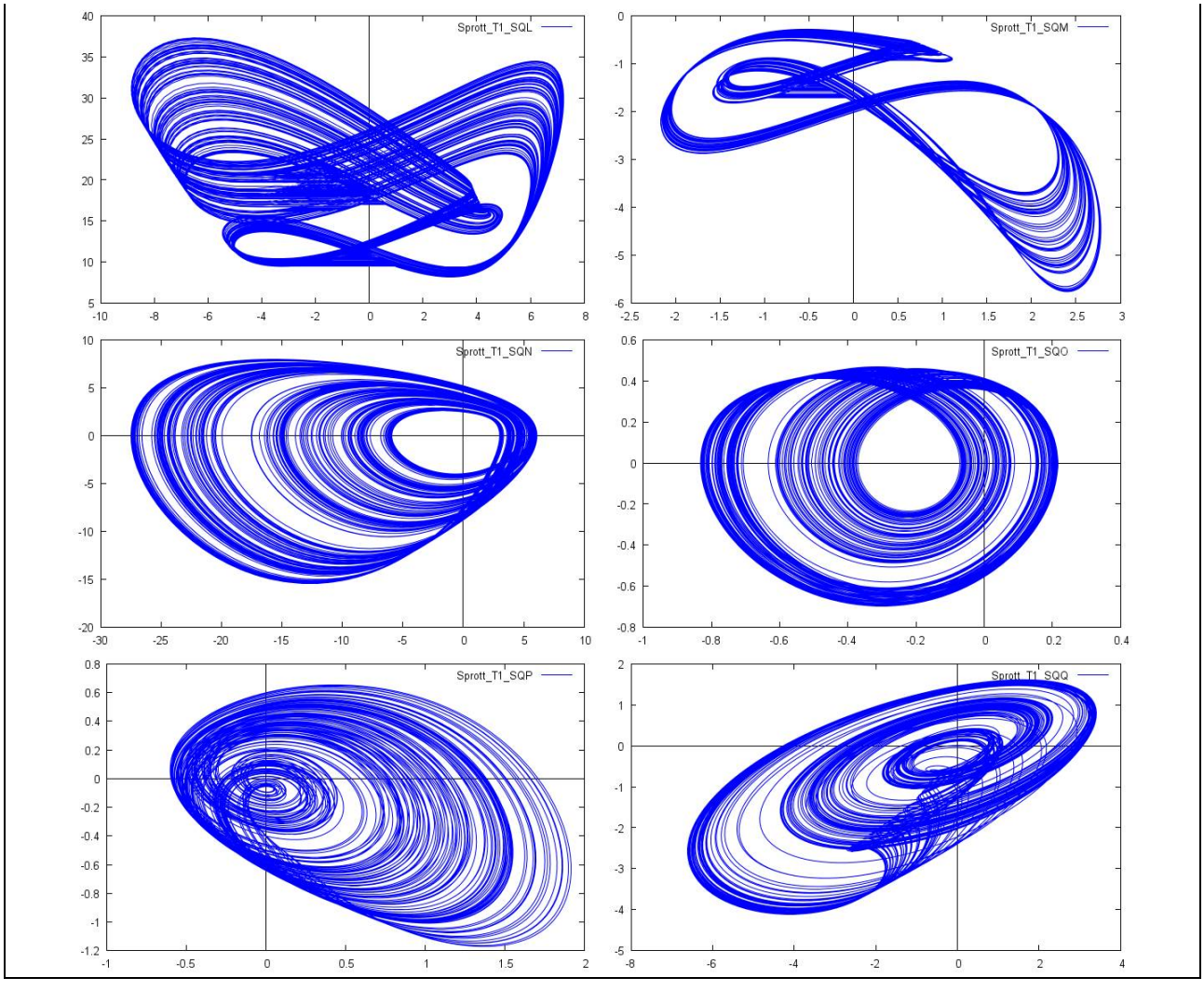


Фигура 7.10. Система на Рьослер: прототип-4.

Име на модела (по Sprott, [165])	Уравнения	x_0, y_0, z_0
SQ_B	$\dot{x} = yz, \dot{y} = x - y, \dot{z} = 1 - xy$	0.6, 0, 0
SQ_C	$\dot{x} = yz, \dot{y} = x - y, \dot{z} = 1 - x^2$	0. 0.5, 0
SQ_D	$\dot{x} = -y, \dot{y} = x + z, \dot{z} = xz + 3y^2$	-0.2, 0, 1
SQ_E	$\dot{x} = yz, \dot{y} = x^2 - y, \dot{z} = 1 - 4x$	-1, 1, 0
SQ_F	$\dot{x} = x + z, \dot{y} = -x + -0.5y, \dot{z} = x^2 - z$	0, 0.2, 0
SQ_G	$\dot{x} = 0.4x + z, \dot{y} = xz - y, \dot{z} = -x + y$	1, 0, 0
SQ_H	$\dot{x} = -y + z^2, \dot{y} = x + 0.5y, \dot{z} = x - z$	0.1, 0, 0
SQ_I	$\dot{x} = -0.2y, \dot{y} = x + z, \dot{z} = x + y^2 - z$	0, 0.3, 0
SQ_J	$\dot{x} = 2z, \dot{y} = -2y + z, \dot{z} = -x + y + y^2$	0, 1, 4
SQ_K	$\dot{x} = xy - z, \dot{y} = x - y, \dot{z} = x + 0.3z$	1, 1, 2
SQ_L	$\dot{x} = y + 3.9z, \dot{y} = 0.9x^2 - y, \dot{z} = 1 - x$	0, 12, -6
SQ_M	$\dot{x} = -z, \dot{y} = -x^2 - y, \dot{z} = 1.7 + 1.7x + y$	1, -0.8, 0
SQ_N	$\dot{x} = -2y, \dot{y} = x + z^2, \dot{z} = 1 + y - 2z$	4.5, 1, 0
SQ_O	$\dot{x} = y, \dot{y} = x - z, \dot{z} = x + xz + 2.7y$	0, 0, 0.5
SQ_P	$\dot{x} = 2.7y + z, \dot{y} = -x + y^2, \dot{z} = x + y$	0, 0.3, 0
SQ_Q	$\dot{x} = -z, \dot{y} = x - y, \dot{z} = 3.1 + y^2 + 0.5z$	1, 0, 0

Таблица 7.1. Някои примери на прости системи с квадратични членове, притежаващи хаотично поведение.


Sprott_T1_SQB
Sprott_T1_SQC
Sprott_T1_SQD
Sprott_T1_SQE
Sprott_T1_SQF
Sprott_T1_SQG
Sprott_T1_SQH
Sprott_T1_SQI
Sprott_T1_SQK



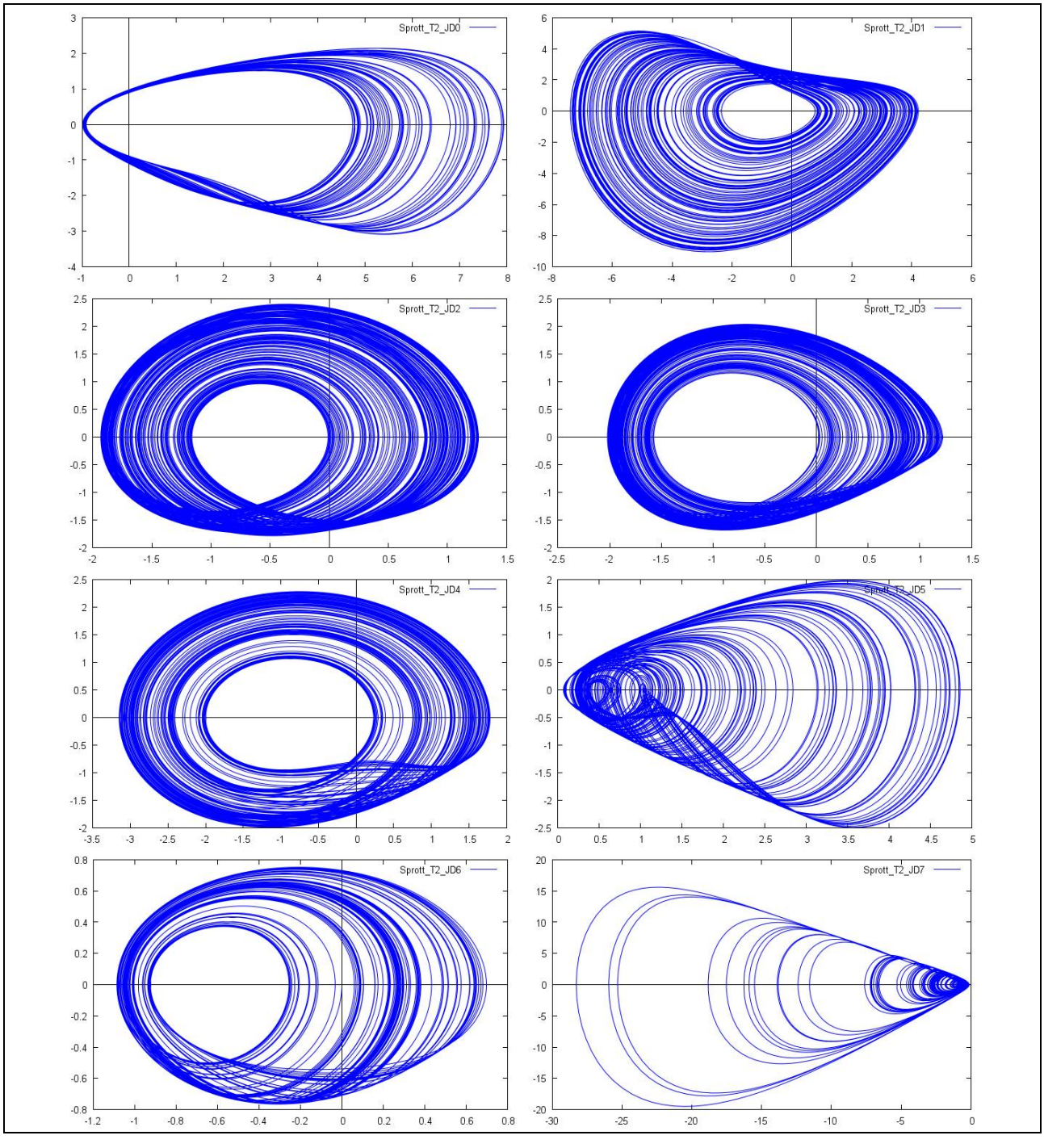
Фигура 7.11. Илюстрации за хаотичното поведение някои прости системи с квадратични членове.

Системи с "тласък" (jerk systems). Такива са с-мите, които могат да се опишат с уравнение от вида $\dddot{x} = f\left(x, \dot{x}, \ddot{x}\right)$. За подобни с-ми, в които за отместване x, $\dot{x}$ и $\ddot{x}$ са съответно скоростта и ускорението, $\dddot{x}$ се нарича „тласък" (jerk). Това е най-простото ОДУ, което би могло да има хаотично поведение. Добре известно е, че всяко ОДУ в явна форма (решено спрямо висшата производна) може да се представи като с-ма ОДУ от първи ред. Обратното не е вярно в общия случай. (Впрочем системите на Лоренц и Рьослер се представят като с-ми с тласък, но видът е много по-тромав, може да се види в споменатата книга на Sprott [165]). Следват няколко примера много прости с-ми с тласък, които имат хаотично поведение (всичките имат „един дял на атрактора").

Име на модела (по Sprott, [165])	Уравнение	x_0, y_0, z_0
JD_0	$\dddot{x} = -2.02\ddot{x} + \dot{x}^2 - x$	5, 2, 0
JD_1	$\dddot{x} = -1.8\ddot{x} - 2x + x\dot{x} - 1$	-5, 5, 0.9
JD_2	$\dddot{x} = -0.4\ddot{x} - 2.1x + x^2 - 1$	0, 2, -1

JD_3	$\dddot{x} = -0.6\ddot{x} - \dot{x} + 0.9x^2 + x\dot{x} - 1$	-2, 0, 3
JD_4	$\dddot{x} = -\ddot{x} - 0.7\dot{x} + x^2 + x\dot{x} - 1$	-2, 0, 1
JD_5	$\dddot{x} = -\ddot{x} + 3\dot{x}^2 - x^2 - x\dot{x}$	1, 0.2, 0
JD_6	$\dddot{x} = -\ddot{x} - \dot{x} + 2x^2 + 2\dot{x}^2 + x\dot{x} - 1$	0, 0, -0.7
JD_7	$\dddot{x} = -\ddot{x} + \dot{x} + 2x^2 - 3\dot{x}^2 + x\dot{x} + x\ddot{x} - 1$	-0.7, 0, 0

Таблица 7.2. Примери за системи с тласък.

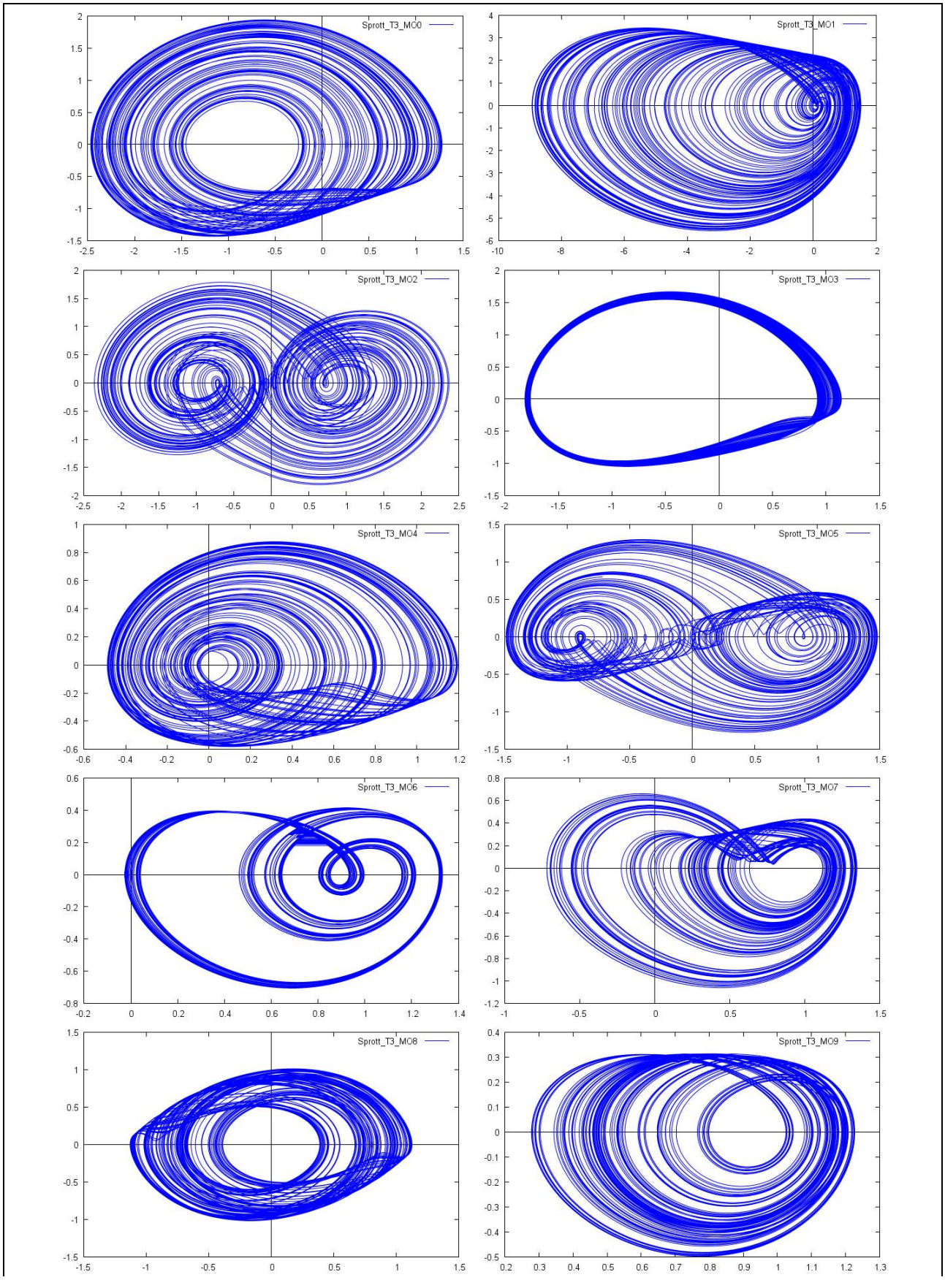


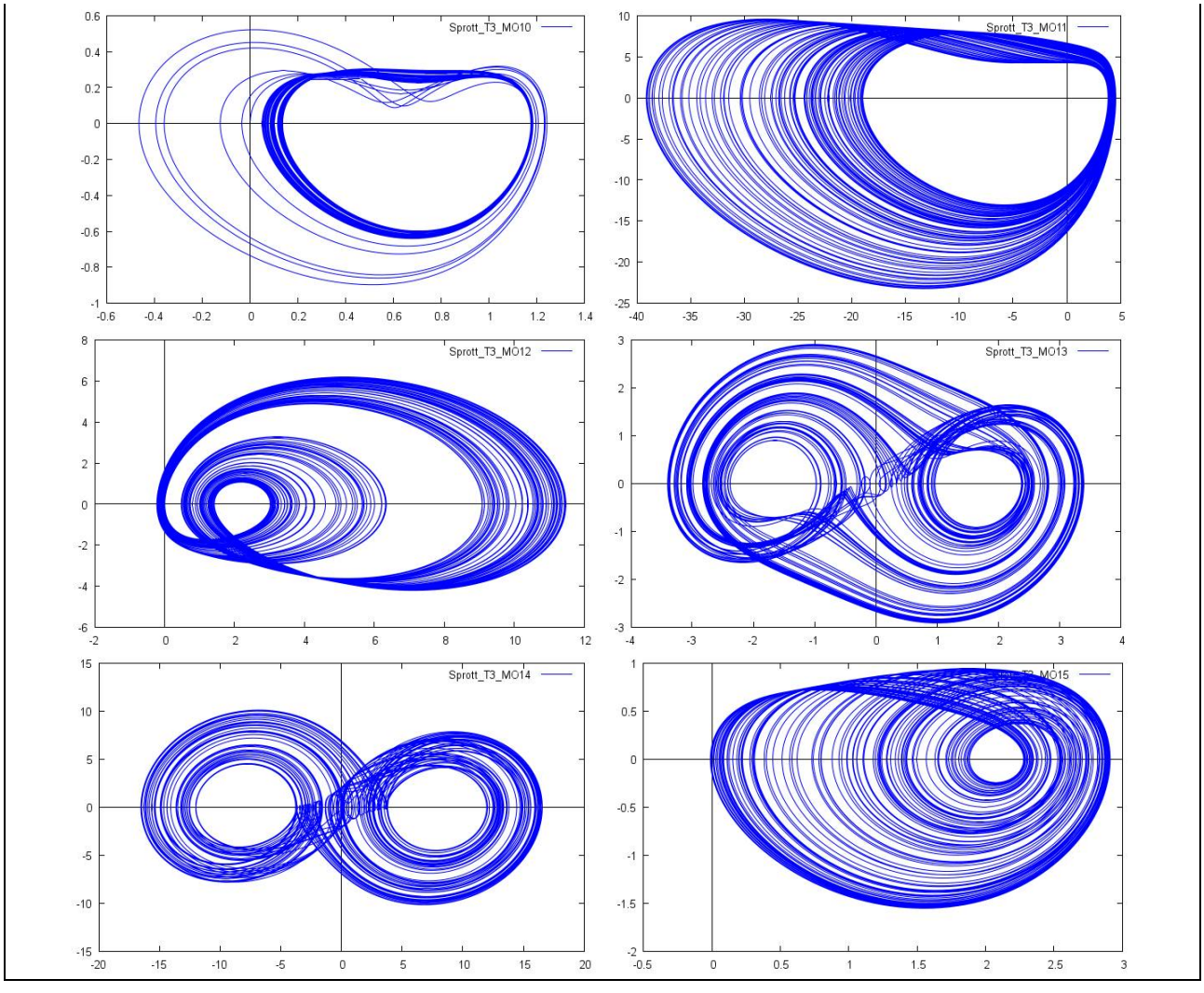
Фигура 7.12. Илюстрации за хаотичното поведение няколко прости системи с тласък.

Осцилатори с памет. Това са системи с тласък, където уравнението се записва като $\dddot{x} + a\ddot{x} + \dot{x} = g(x)$. Следват няколко примера за подобни системи, които са с хаотично поведение.

Име на модела (по Sprott, [165])	Уравнение	x_0, y_0, z_0
MO_0	$\dddot{x} + 0.6\ddot{x} + \dot{x} = \lvert x \rvert - 1$	0, -0.7, 0
MO_1	$\dddot{x} + 0.6\ddot{x} + \dot{x} = 1 - 6\max(x, 0)$	0, 0, 0.5
MO_2	$\dddot{x} + 0.6\ddot{x} + \dot{x} = \mathrm{sgn}(x) - x$	0, 0, 1
MO_3	$\dddot{x} + \ddot{x} + \dot{x} = 1.1\left(x^2 - 1\right)$	1, 0, -1
MO_4	$\dddot{x} + 0.5\ddot{x} + \dot{x} = x\left(x - 1\right)$	0, 0.1, 0
MO_5	$\dddot{x} + 0.7\ddot{x} + \dot{x} = x\left(1 - x^2\right)$	0, 0, 0.1
MO_6	$\dddot{x} + 0.4\ddot{x} + \dot{x} = x^2\left(1 - x\right)$	0, 0, 0.5
MO_7	$\dddot{x} + 0.6\ddot{x} + \dot{x} = x^2\left(1 - x^2\right)$	0, 0, 0.4
MO_8	$\dddot{x} + 0.5\ddot{x} + \dot{x} = x\left(x^4 - 1\right)$	0, 0.9, 0
MO_9	$\dddot{x} + 0.4\ddot{x} + \dot{x} = x^3\left(1 - x\right)$	1. 0.2, 0
MO_{10}	$\dddot{x} + 0.6\ddot{x} + \dot{x} = x^2\left(1 - x^3\right)$	0, 0, 0.4
MO_{11}	$\dddot{x} + \ddot{x} + \dot{x} = 5 - e^x$	0, 4.3, 0
MO_{12}	$\dddot{x} + 0.5\ddot{x} + \dot{x} = 7 - 8\tanh(x)$	0, 1, 6
MO_{13}	$\dddot{x} + \ddot{x} + \dot{x} = 6\tanh(x) - 3x$	0, 0, 1
MO_{14}	$\dddot{x} + 0.6\ddot{x} + \dot{x} = 6\arctan(x) - x$	0, 1, 6
MO_{15}	$\dddot{x} + 0.6\ddot{x} + \dot{x} = x - 0.5\sinh(x)$	0, 0, 1

Таблица 7.3. Примери на осцилатори с памет.


Sprott_T3_MO0
Sprott_T3_MO1
Sprott_T3_MO2
Sprott_T3_MO3
Sprott_T3_MO4
Sprott_T3_MO5
Sprott_T3_MO6
Sprott_T3_MO7
Sprott_T3_MO8
Sprott_T3_MO9

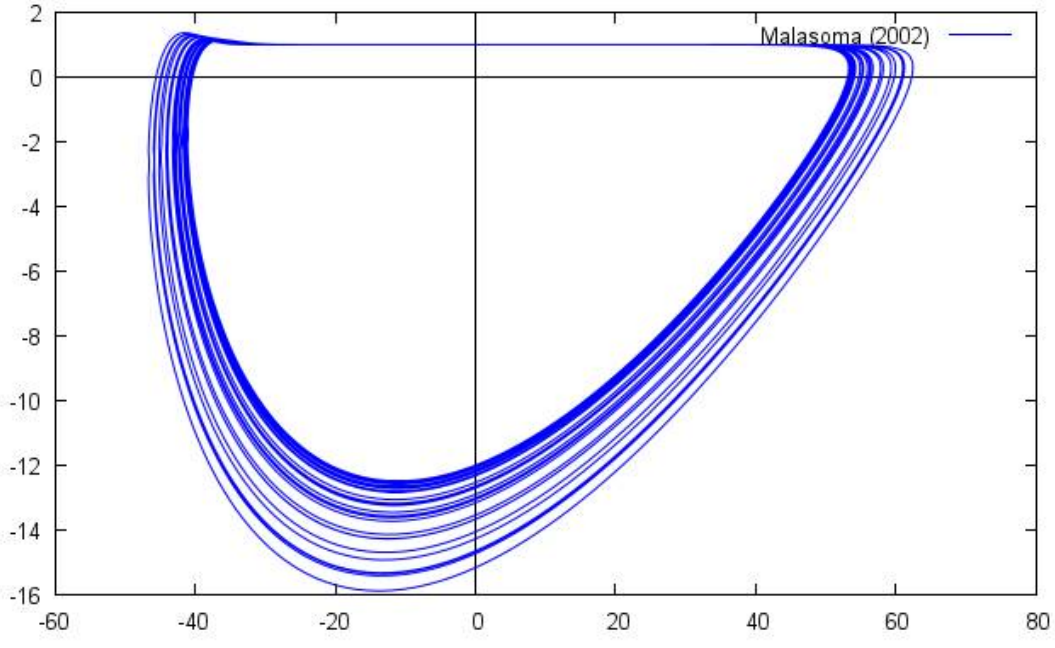


Фигура 7.13. Илюстрации за хаотичното поведение няколко прости системи от тип "осцилатор с памет".

Система, която се свежда от система с тласък с рационална нелинейност [166]. Системата

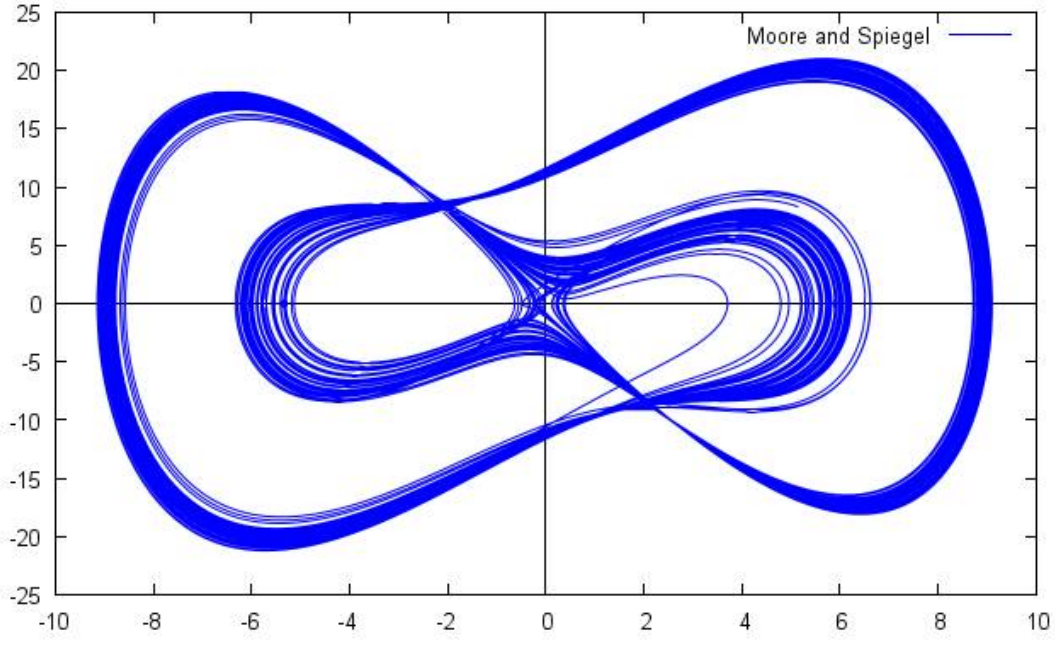
$$\begin{cases} \dot{x} = z \\ \dot{y} = -\alpha y + z \\ \dot{z} = -x + xy \end{cases} \tag{7.5}$$

има само 5 члена и една нелинейност и може да се приведе във вида $\dddot{x} = -\alpha x - \alpha \ddot{x} + x\dot{x} + \dot{x}\ddot{x} / x$. За $\alpha = 10.3$ има хаотично поведение. Илюстрацията по-долу е с начални стойности $(x_0, y_0, z_0) = (46,1,10)$.



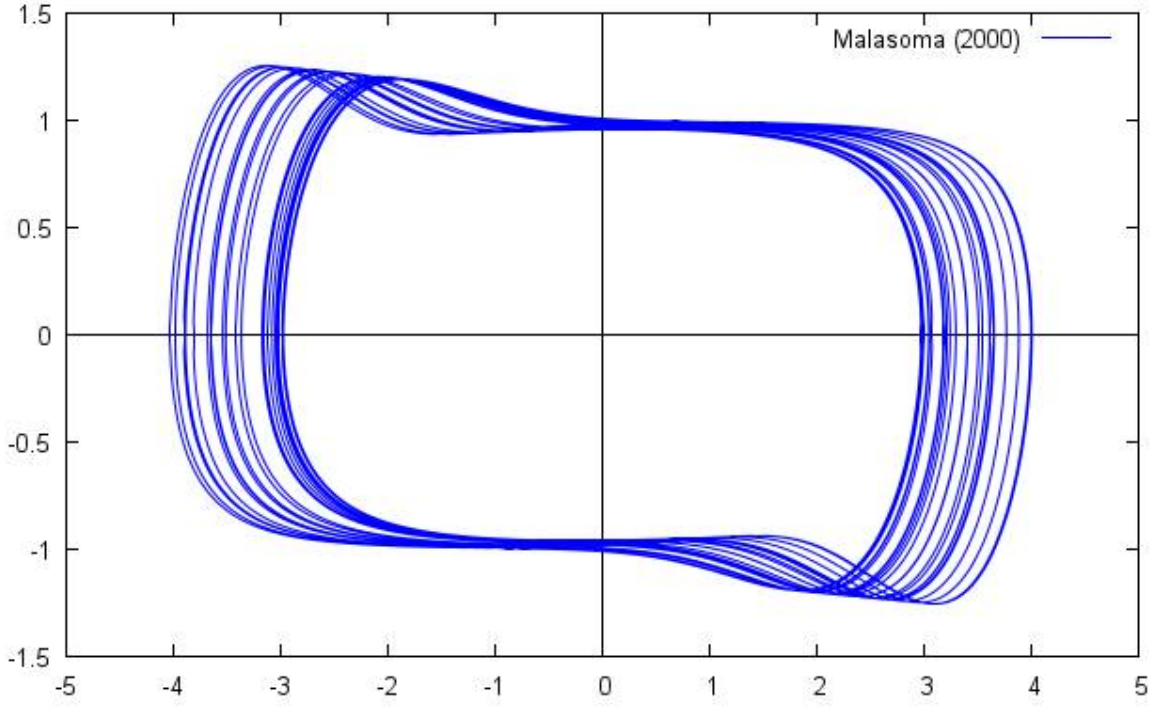
Фигура 7.14. Система, която се свежда до система с тласък с една рационална нелинейност: $\dddot{x} = -\alpha x - \alpha \ddot{x} + x\dot{x} + \dot{x}\ddot{x} / x$, за стойност на параметъра $\alpha = 10.3$, при която системата има хаотично поведение.

Система с тласък с кубична нелинейност. За такъв тип системи са известни доста случаи на хаотично поведение. Една такава, известна още от 1966 г. [167] се описва с уравнението $\dddot{x} = -\ddot{x} + 9\dot{x} - x^2\dot{x} - 5x$. Илюстрацията е за начални стойности $(x_0, y_0, z_0) = (4,7,2)$.



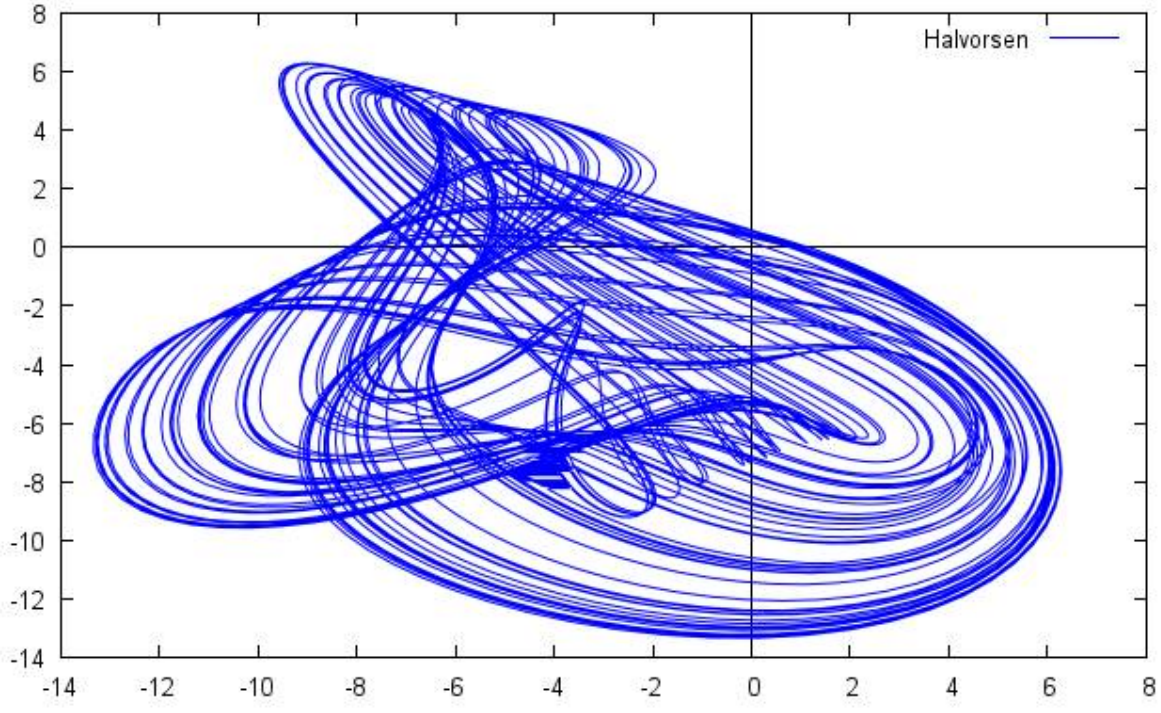
Фигура 7.15. Илюстрации за хаотичното поведение на система с тласък с кубичен член: $\dddot{x} = -\ddot{x} + 9\dot{x} - x^2\dot{x} - 5x$.

Най-простият вид на подобни система с хаотично поведение се описва с уравнението $\dddot{x} = -a\ddot{x} + x^2\dot{x} - x$. Илюстрацията е за $a = 2.03$ и начални стойности $(x_0, y_0, z_0) = (0, 0.96, 0)$.



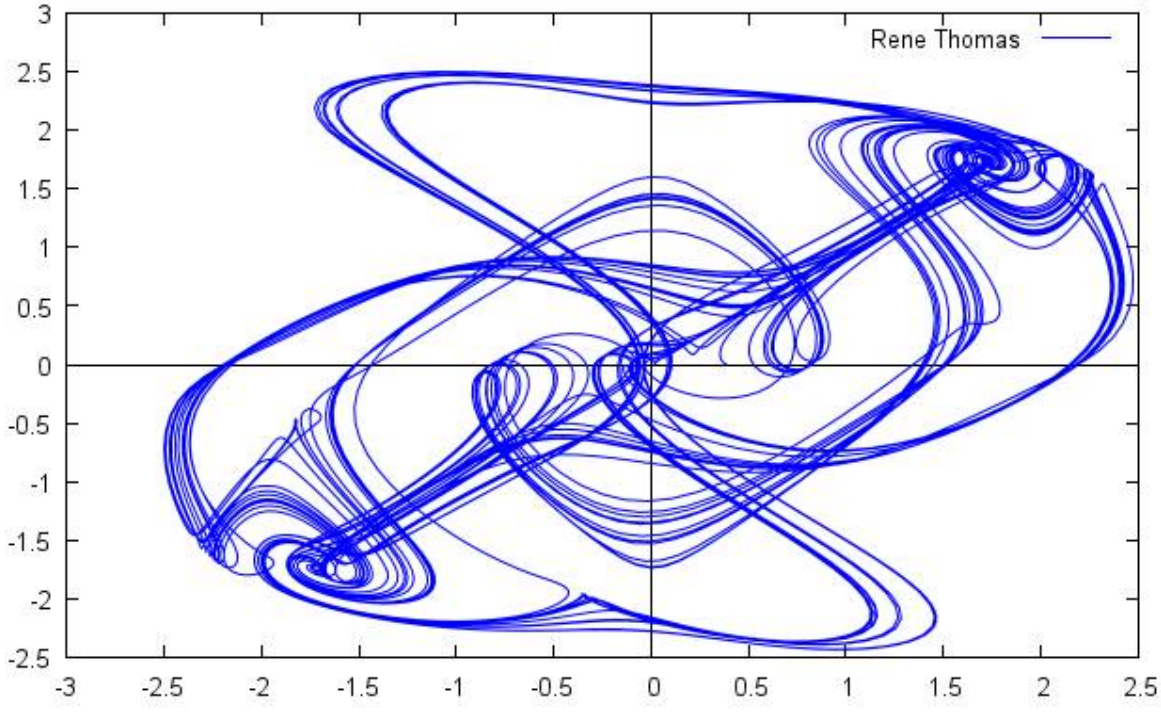
Фигура 7.16. Илюстрации на най-простия вид система с тласък с кубичен член: $\dddot{x} = -a\ddot{x} + x^2\dot{x} - x$. При $a = 2.03$ поведението е хаотично.

Циклична система. Това е система, която се описва с една и съща функция $f(x, y, x)$ в десните части, но аргументите се променят циклично $(x, y, z) \to (y, z, x) \to (z, x, y)$. Приведеният пример е на Arne Dehli Halvorsen, публикуван за първи път в [165]: $f(x, y, x) = -ax - 4y - 4z - y^2$. Илюстрацията е за $a = 1.3$ и $(x_0, y_0, z_0) = (-6.4, 0, 0)$.



Фигура 7.17. Цикличната система на Халворсен.

Цикличната система с кубична нелинейност [168]. Системата има кубични нелинейности и се описва с $f(x,y,x) = -ax + by - y^3$. За илюстрацията $a = 1, b = 4$ и $(x_0, y_0, z_0) = (0.4, 0, 0)$.



Фигура 7.18. Циклична система с кубична нелинейност:
$f(x,y,x) = -ax + by - y^3$.

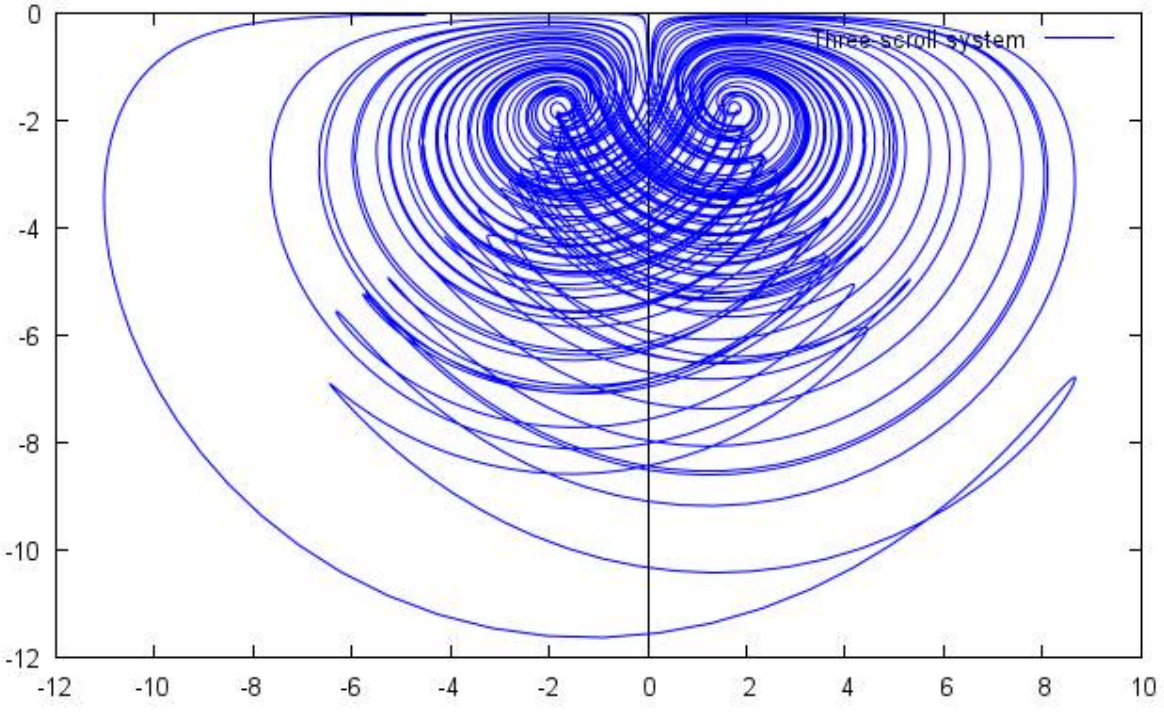
Системи с няколко дяла на атрактора. Едни от най-простите (само с квадратични нелинейности) са

$$\begin{cases} \dot{x} = x - yz \\ \dot{y} = -y + xz \\ \dot{z} = -3z + xy \end{cases} \tag{7.6}$$

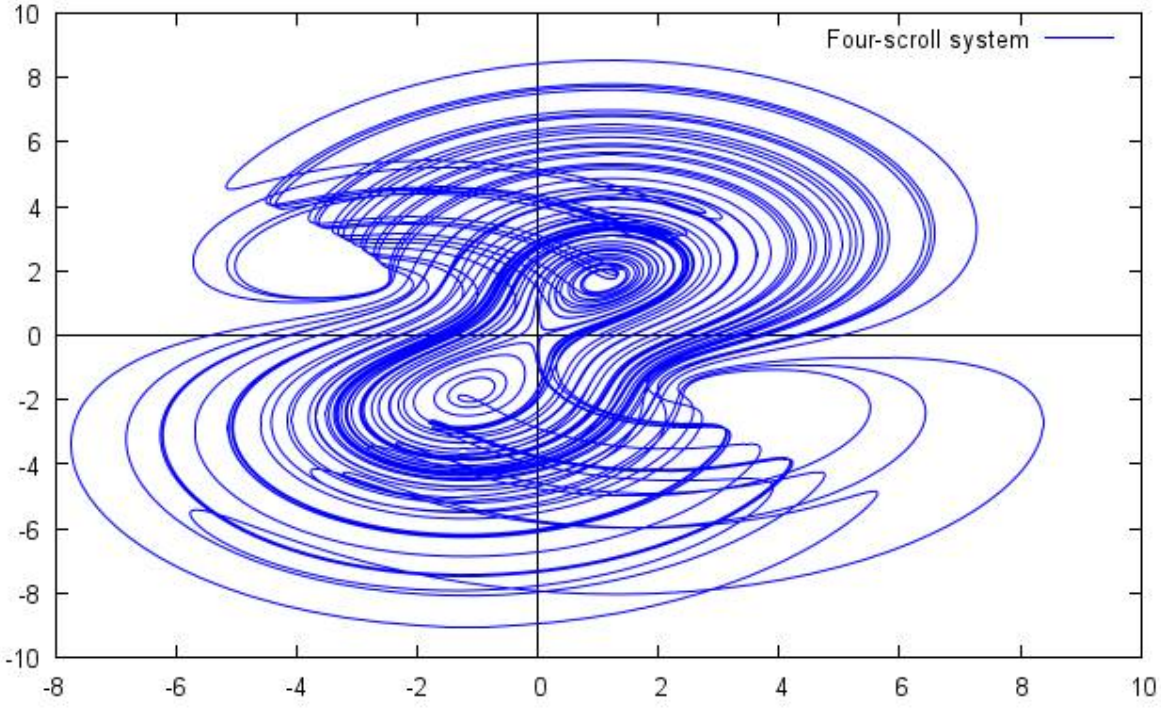
и

$$\begin{cases} \dot{x} = x - yz \\ \dot{y} = x - y + xz \\ \dot{z} = -3z + xy. \end{cases} \tag{7.7}$$

Илюстрациите са за начални условия съответно $(x_0, y_0, z_0) = (1, -2, 0)$ и $(x_0, y_0, z_0) = (1, 0, 0)$.



Фигура 7.19. Системата (5) с два дяла на атрактора.

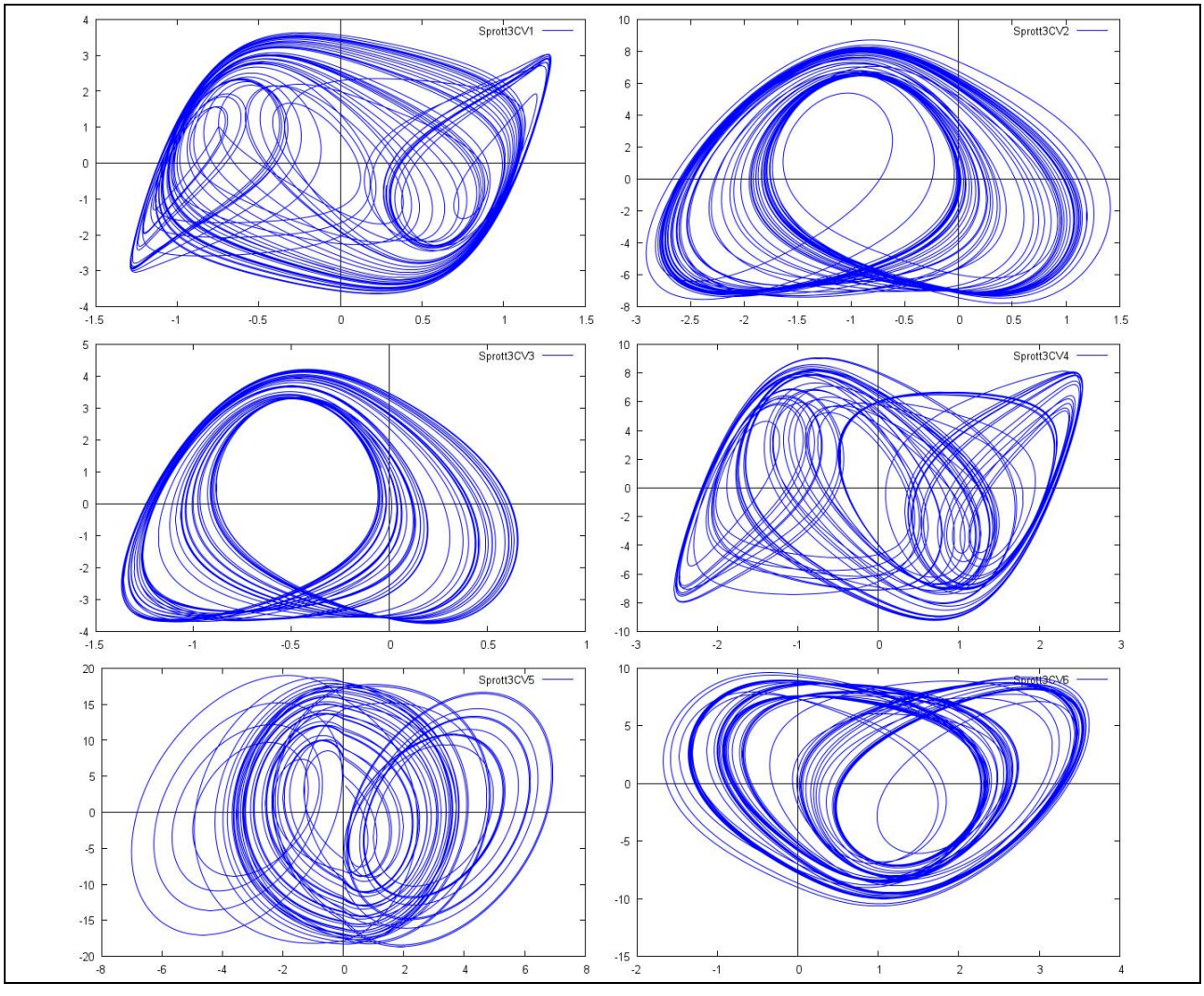


Фигура 7.20. Системата (6) с два дяла на атрактора.

Системи на Чуа [169]. В таблицата са дадени няколко примера.Приведено само съответното първо уравнение. Второто и третото уравнение за всички системи са съответно $\dot{y} = x + z$ и $\dot{z} = -y$.

Име на модела (по Sprott, [165])	Първо уравнение	x_0, y_0, z_0
CV_1	$\dot{x} = 0.3y + x - x^3$	0, -3, 1
CV_2	$\dot{x} = 0.2y - x + 2\tanh(x)$	0, 1, -6
CV_3	$\dot{x} = 0.2y - x - x\lvert x\rvert$	0, 1, -3
CV_4	$\dot{x} = 0.2y - x + 2\sin(x)$	0, 6, 0
CV_5	$\dot{x} = 0.2y - 0.3x + \mathrm{sgn}(x)$	0, 4, -8
CV_6	$\dot{x} = 0.2y - x + 2\arctan(x)$	0.7, 6, 0

Таблица 7.4. Премери за системи на Чуа.

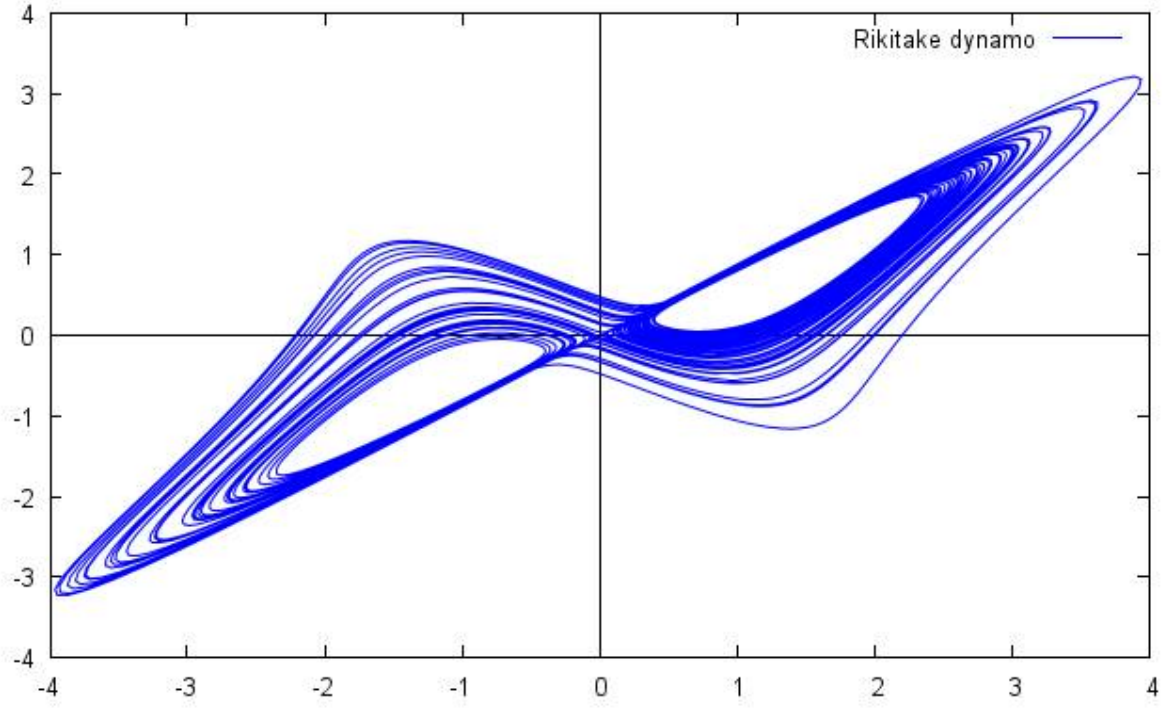


Фигура 7.21. Илюстрации за хаотично поведение на системи на Чуа.

Динамото на Рикитаке [170]. Системата е

$$\begin{cases} \dot{x} = -\mu x + yz, \\ \dot{y} = -\mu y + x\left(z - \alpha\right), \\ \dot{z} = 1 - xy. \end{cases} \tag{7.8}$$

Илюстрацията е за $\mu = \alpha = 1$ и начални условия $\left(x_0, y_0, z_0\right) = \left(1, 0, 0.6\right)$.



Фигура 7.22. Динамо на Рикитаке.

7.3. Заключение.

Приведени са примерни резултати от практическо решаване с висока точност на системи с "хаотично поведение", получени с разработените в настоящата работа програмни инструменти. Първата част е илюстрация към обсъждането в раздел 3.1 от глава 3 за възможностите за количествено изследване при решаване с все по-голяма точност. Останалите примери са илюстрации на решения на прости гладки динамични системи с хаотично поведение, които, освен естетическата си стойност, послужиха и за допълнителни тестове за предлаганите средства за решаване в настоящата програмна система.

Глава 8. Приложения свързани с бързото пресмятане на някои математически константи.

8.1. Пресмятане на константата на Апери с разпаралеляване по метода на двоичното разделяне.

```
// zeta(3), Apery
internal class series_ap_a
{
   internal mpir.mpz_t res;
   internal mpir.mpz_t this[int n]
   {
      get
      {
         mpir.mpz_set_si(res, n);
         mpir.mpz_mul_ui(res, res, 205);
         mpir.mpz_add_ui(res, res, 250);
         mpir.mpz_mul_si(res, res, n);
         mpir.mpz_add_ui(res, res, 77);
         if (n % 2 == 1) mpir.mpz_neg(res, res);
         return res;
      }
   }
   public series_ap_a()
   {
      res = mpir.mpz_init();
   }
}
internal class series_ap_p
{
   internal mpir.mpz_t res;
   internal mpir.mpz_t this[int n]
   {
      get
      {
         if (n > 0)
         {
            mpir.mpz_set_si(res, n);
            mpir.mpz_pow_ui(res, res, 10);
         }
         else mpir.mpz_set_ui(res, 1);
         return res;
      }
   }
   public series_ap_p()
   {
      res = mpir.mpz_init();
```

```
    }
}
internal class series_ap_q
{
    private mpir.mpz_t res;
    internal mpir.mpz_t this[int n]
    {
        get
        {
            if (n > 0)
            {
                mpir.mpz_set_si(res, 2 * n);
                mpir.mpz_mul_si(res, res, 2 * n + 1);
                mpir.mpz_pow_ui(res, res, 5);
            }
            else mpir.mpz_set_ui(res, 1);
            return res;
        }
    }
    public series_ap_q()
    {
        res = mpir.mpz_init();
    }
}
internal class series_ap
{
    public series_ap_a a;
    public series_ap_p p;
    public series_ap_q q;

    public series_ap()
    {
        a = new series_ap_a();
        p = new series_ap_p();
        q = new series_ap_q();
    }
}

public class series_ap_result
{
    internal mpir.mpz_t P, Q, T; // B not needed
    public series_ap_result()
    {
        P = mpir.mpz_init();
        Q = mpir.mpz_init();
        T = mpir.mpz_init();
    }
}
```

```
namespace MPConsts_R
{
    public class MPConsts
    {
        ......
        static void sum_ap(series_ap_result r, int n1, int n2, series_ap arg)
        {
            mpir.mpz_t tmp = mpir.mpz_init();
            switch (n2 - n1)
            {
                case 0:
                    Console.WriteLine("sum_apq: n2-n1 should be > 0.");
                    break;

                case 1:
                    mpir.mpz_set(r.P, arg.p[n1]);
                    mpir.mpz_set(r.Q, arg.q[n1]);
                    mpir.mpz_mul(r.T, arg.a[n1], arg.p[n1]);
                    break;

                case 2:
                    mpir.mpz_set(r.P, arg.p[n1]);
                    mpir.mpz_mul(r.P, r.P, arg.p[n1 + 1]);

                    mpir.mpz_set(r.Q, arg.q[n1]);
                    mpir.mpz_mul(r.Q, r.Q, arg.q[n1 + 1]);

                    // T = BQS = a[n1]p[n1]b[n1+1]q[n1+1] + a[n1+1]p[n1]p[n1+1]b[n1]
                    mpir.mpz_mul(r.T, arg.a[n1], arg.p[n1]);

                    mpir.mpz_mul(tmp, arg.a[n1 + 1], r.P);
                    mpir.mpz_add(r.T, r.T, tmp);
                    break;

                // case 3, 4 skipped

                default: // general case
                    // the left and the right partial sum
                    series_ap_result L = new series_ap_result();
                    series_ap_result R = new series_ap_result();
                    // find the middle of the interval
                    int nm = (n1 + n2) / 2;
                    // sum left side
                    sum_ap(L, n1, nm, arg);
                    // sum right side
                    sum_ap(R, nm, n2, arg);
                    // put together
```

```
                mpir.mpz_mul(r.P, L.P, R.P);
                mpir.mpz_mul(r.Q, L.Q, R.Q);

                mpir.mpz_mul(r.T, R.Q, L.T);
                mpir.mpz_mul(tmp, L.P, R.T);

                mpir.mpz_add(r.T, r.T, tmp);
                break;
            }
        }

        internal static series_ap_result combine2ap(series_ap_result r1,
series_ap_result r2)
        {
            series_ap_result rall = new series_ap_result();
            mpir.mpz_t tmp = mpir.mpz_init();
            mpir.mpz_mul(rall.P, r1.P, r2.P);
            mpir.mpz_mul(rall.Q, r1.Q, r2.Q);

            mpir.mpz_mul(rall.T, r2.Q, r1.T);
            mpir.mpz_mul(tmp, r1.P, r2.T);
            mpir.mpz_add(rall.T, rall.T, tmp);
            return rall;
        }
        internal static series_ap_result partial_sum_ap(int a, int b)
        {
            series_ap arg = new series_ap();
            series_ap_result r = new series_ap_result();

            sum_ap(r, a, b, arg);

            return r;
        }

        internal static mpir.mpf_t apery(uint digits)
        {
            uint prec0 = (uint)(digits / Math.Log10(2)) + 1;
            //uint prec = prec0 + 10;
            uint prec = prec0 + (uint)Math.Max(10, Math.Log(digits) + 1);

            int nmax = (int)(0.1 * digits * Math.Log(10) / Math.Log(2));
            int cnt = Environment.ProcessorCount;
            int chunk = nmax / cnt;

            var tasks = new List<Task<series_ap_result>>();
            for (int n = 0; n < cnt; n++)
            {
                int idx = n;
```

```
    int idx1 = chunk * idx;
    int idx2 = chunk * (idx + 1); if (n == cnt - 1) idx2 = nmax;
    //Console.WriteLine("idx1 = {0}, idx2 = {1}", idx1, idx2);
    tasks.Add(Task.Factory.StartNew(() =>
    {
        return partial_sum_ap(idx1, idx2);
    }//, TaskCreationOptions.LongRunning
    ));
}

var whenAllTask = Task.WhenAll<series_ap_result>(tasks);
List<series_ap_result> rr = new List<series_ap_result>();
List<series_ap_result> rr1 = new List<series_ap_result>();
series_ap_result rall = new series_ap_result();

var final = whenAllTask.ContinueWith(t =>
{
    for (int i = 0; i < tasks.Count; i++)
    {
        rr.Add(t.Result[i]);
    }
}, TaskContinuationOptions.OnlyOnRanToCompletion
);
final.Wait();

int lim = cnt / 2;
while (lim > 1)
{
    tasks.Clear();
    tasks = new List<Task<series_ap_result>>();
    for (int i = 0; i < lim; i++)
    {
        int idx = i;
        tasks.Add(Task.Factory.StartNew(() =>
        {
            return combine2ap(rr[2 * idx], rr[2 * idx + 1]);
        }//, TaskCreationOptions.LongRunning
    ));
    }
    rr1.Clear();
    whenAllTask = Task.WhenAll<series_ap_result>(tasks);
    var final1 = whenAllTask.ContinueWith(t =>
    {
        for (int i = 0; i < tasks.Count; i++)
        {
            rr1.Add(t.Result[i]);
        }
    }, TaskContinuationOptions.OnlyOnRanToCompletion
```

```
            );
            final1.Wait();
            rr.Clear();
            rr.AddRange(rr1);
            lim /= 2;
        }
        rall = combine2ap(rr[0], rr[1]);
        mpir.mpf_t sum = mpir.mpf_init2(prec); mpir.mpf_set_z(sum, rall.T);
        mpir.mpf_t K = mpir.mpf_init2(prec); mpir.mpf_set_z(K, rall.Q);

        mpir.mpf_div(sum, sum, K);
        mpir.mpf_div_2exp(sum, sum, 6);

        return sum;
    }
    .....
  }
}
```

Фигура 8.1. Примерен код за пресмятане на математическа константа с разпаралеляване.

8.2. Някои редове, подходящи за пресмятане по метода на двоичното разделяне (binary splitting).

За константа на Apéry $\zeta(3)=\sum_{n=1}^{\infty}\frac{1}{n^3}$

$$(8.1)\qquad \begin{cases} \zeta(3)=\frac{1}{4}\sum_{n=1}^{\infty}(-1)^{n-1}\frac{56n^2-32n+5}{n^3(2n-1)^2}\frac{(n!)^4}{(3n)!(2n)!}, \\ \qquad =\frac{1}{4}\sum_{n=1}^{\infty}(-1)^{n-1}\frac{56n^2-32n+5}{n^3(2n-1)^2}\frac{1}{\binom{3n}{n}\binom{2n}{n}}, \\ \text{T. Amdeberhan, 1996,[6]} \end{cases}$$

$$(8.2)\qquad \begin{cases} \zeta(3)=\frac{1}{72}\sum_{n=0}^{\infty}(-1)^{n}\frac{5265n^4+13878n^3+13761n^2+6120n+1040}{(4n+1)(4n+3)(n+1)(3n+1)^2(3n+2)^2}\frac{(2n)!(n!)^2}{(4n)!}, \\ \qquad =\frac{1}{72}\sum_{n=0}^{\infty}(-1)^{n}\frac{5265n^4+13878n^3+13761n^2+6120n+1040}{(4n+1)(4n+3)(n+1)(3n+1)^2(3n+2)^2}\frac{1}{\binom{4n}{n}\binom{3n}{n}}, \\ \text{T. Amdeberhan, 1996,[6]} \end{cases}$$

(8.3)
$$\begin{cases} \zeta(3)=\dfrac{1}{2}\displaystyle\sum_{n=0}^{\infty}(-1)^n\left(205n^2+250n+77\right)\dfrac{\left((n+1)!\right)^5(n!)^5}{\left((2n+2)!\right)^5}, \\ \quad =\dfrac{1}{64}\displaystyle\sum_{n=0}^{\infty}(-1)^n\left(205n^2+250n+77\right)\dfrac{(n!)^{10}}{\left((2n+1)!\right)^5}, \\ \text{T. Amdeberhan, D. Zeilberger, 1997,[7]} \end{cases}$$

(8.4)
$$\begin{cases} \zeta(3)=\dfrac{2}{5}\displaystyle\sum_{n=1}^{\infty}(-1)^{n-1}\frac{1}{n^3}\frac{(n!)^2}{(2n)!}=\frac{5}{2}\sum_{n=1}^{\infty}(-1)^{n-1}\frac{1}{n^3}\frac{1}{\binom{2n}{n}}, \\ \text{M.}-\text{P. Chen, H. Srivastava, 1998,[43]} \end{cases}$$

(8.5)
$$\begin{cases} \zeta(3)=\dfrac{1}{24}\displaystyle\sum_{n=0}^{\infty}(-1)^n\,a(n)\frac{\left((2n+1)!(2n)!n!\right)^3}{(3n+2)!\left((4n+3)!\right)^3}, \\ a(n)=126392n^5+412708n^4+531578n^3+336367n^2+104000n+12463, \\ \text{S. Wedeniwski, 1998,[141]} \end{cases}$$

За константа на Catalan $G=\beta(2)=\displaystyle\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)^2}$

(8.6)
$$\begin{cases} G=\dfrac{\pi}{8}\ln\left(2+\sqrt{3}\right)+\dfrac{3}{8}\displaystyle\sum_{n=1}^{\infty}\frac{1}{(2n+1)^2}\frac{(n!)^2}{(2n)!}, \\ \quad =\dfrac{\pi}{8}\ln\left(2+\sqrt{3}\right)+\dfrac{3}{8}\displaystyle\sum_{n=1}^{\infty}\frac{1}{(2n+1)^2}\frac{1}{\binom{2n}{n}}, \\ \text{S. Ramanujan, 1915,[114]} \end{cases}$$

(8.7)
$$\begin{cases} G=-\dfrac{1}{64}\displaystyle\sum_{n=1}^{\infty}(-1)^n\frac{40n^2-24n+3}{n^3(2n-1)}\frac{2^{8n}(2n)!(n!)^2}{(4n)!}, \\ \quad =-\dfrac{1}{64}\displaystyle\sum_{n=1}^{\infty}(-1)^n\frac{40n^2-24n+3}{n^3(2n-1)}\frac{2^{8n}}{\binom{2n}{n}\binom{4n}{2n}}, \\ \text{A. Lupas, 2000,[92]} \end{cases}$$

(8.8)
$$\begin{cases} G = -\frac{1}{2}\sum_{n=1}^{\infty}(-1)^n \frac{3n-1}{n^3}\frac{8^n (n!)^6}{((2n)!)^3}, \\ \quad = -\frac{1}{2}\sum_{n=1}^{\infty}(-1)^n \frac{3n-1}{n^3}\frac{8^n}{\binom{2n}{n}^3}, \\ \text{Z. – W. Sun, 2011,[132]} \end{cases}$$

(8.9)
$$\begin{cases} G = \frac{1}{2}\sum_{n=0}^{\infty}(-1)^n \frac{3n+2}{(2n+1)^3}\frac{8^n (n!)^6}{((2n)!)^3}, \\ \quad = \frac{1}{2}\sum_{n=0}^{\infty}(-1)^n \frac{3n+2}{(2n+1)^3}\frac{8^n}{\binom{2n}{n}^3}, \\ \text{F. Lima, 2012,[88]} \end{cases}$$

За π

(8.10)
$$\begin{cases} \frac{\pi^2}{6} = 3\sum_{n=1}^{\infty}\frac{1}{n^2}\frac{(n!)^2}{(2n)!} = 3\sum_{n=1}^{\infty}\frac{1}{n^2}\frac{1}{\binom{2n}{n}}, \\ \text{L. Euler, 1748} \end{cases}$$

(8.11)
$$\begin{cases} \frac{1}{\pi} = \sum_{n=1}^{\infty}(42n+5)\frac{(2n)!}{2^{12n+4}(n!)^2} = \sum_{n=0}^{\infty}(42n+5)\frac{\binom{2n}{n}}{2^{12n+4}}, \\ \text{S. Ramanujan 1914, [113]} \end{cases}$$

(8.12)
$$\begin{cases} \frac{1}{\pi} = \frac{\sqrt{8}}{9801}\sum_{n=1}^{\infty}(26390n+1103)\frac{1}{396^n}\frac{(4n)!}{(n!)^4}, \\ \text{S. Ramanujan, 1914,[113]} \end{cases}$$

(8.13)
$$\begin{cases} \frac{\pi^4}{90} = \frac{36}{17}\sum_{n=1}^{\infty}\frac{1}{n^4}\frac{(n!)^2}{(2n)!} = \frac{36}{17}\sum_{n=1}^{\infty}\frac{1}{n^4}\frac{1}{\binom{2n}{n}}, \\ \text{L. Comtet, 1974} \end{cases}$$

(8.14)
$$\begin{cases} \dfrac{\pi}{2} = \displaystyle\sum_{n=0}^{\infty} (25n-3)\frac{n!(2n)!}{2^n (3n)!} = \sum_{n=0}^{\infty} (25n-3)\frac{1}{2^n \binom{3n}{n}}, \\ \text{R. Gosper, 1974, [72]} \end{cases}$$

(8.15)
$$\begin{cases} \dfrac{1}{\pi} = \dfrac{12}{(640320)^{\frac{3}{2}}} \displaystyle\sum_{n=1}^{\infty} (-1)^n \frac{545140134n + 13591409}{640320^{3n}} \frac{(6n)!}{(3n)!(n!)^3}, \\ \text{D.Chudnovsky, G. Chudnovsky, 1989, [45]} \end{cases}$$

(8.16)
$$\begin{cases} \dfrac{\pi^2}{6} = \displaystyle\sum_{n=1}^{\infty} \frac{(21n-8)}{n^3} \frac{(n!)^6}{((2n)!)^3} = \sum_{n=1}^{\infty} \frac{(21n-8)}{n^3} \frac{1}{\binom{2n}{n}^3}, \\ \text{D. Zeilberger, 1993} \end{cases}$$

(8.17)
$$\begin{cases} \dfrac{17\pi^4}{3240} = \displaystyle\sum_{n=1}^{\infty} \frac{1}{n^4} \frac{(n!)^2}{(2n)!} = \sum_{n=1}^{\infty} \frac{1}{n^4} \frac{1}{\binom{2n}{n}} \quad \left[= \frac{17}{37}\zeta(4) \right], \\ \text{D. Bailey, J. Borwein, D. Bradley, 2006} \end{cases}$$

За ln(2)

(8.18)
$$\begin{cases} \ln 2 = \dfrac{2}{3} \displaystyle\sum_{n=0}^{\infty} \frac{1}{(2n+1)} \frac{1}{9^n}, \\ \text{J. M. Borwein, D. H. Bailey, 2003, [28]} \end{cases}$$

8.3. Заключение.

Приведен е типичен за представяната система пример на програмен код за пресмятане на математическа константа, използвайки разпаралеляване на базата на двоично разделяне. Събрани са на едно място представяния на някои константи във вид на редове, които са подходящи за прилагане на двоично разделяне.

Някои възможни насоки за усъвършенстване и разширяване на извършената до момента изследователска дейност.

Възможните насоки за усъвършенстване и разширяване на осъществената до сега изследователска дейност могат най-общо да се групират в три насоки. 1) Добавяне на средства за решаване на нови класове задачи, 2) Добавяне на нови методи за решаване на вече разгледаните класове задачи и 3) усъвършенстване на средствата за решаване. Последната възможност би могла да включва развит интерфейс, даващ повече възможности за детайлизация на изискванията, както и отстъпка в полза на включване на някои "отречени" в досегашната методология спомагателни средства, автоматизиращи конкретни задачи в стил "компютърна алгебра". Някои от тях са относително по-леки за осъществяване, например отделни средства от високо ниво за сумиране на безкрайни редове или автоматично разпознаване на константи. Както беше посочено в увода, основният проблем тук е, че тази насока не може да бъде последователно доведена докрай. Тази насока би могла да включи и специализирането на досега съществуващи средства за решаване за изследване на конкретни интересни проблеми, например изследване на модели на нелинейната динамика, пораждащи "детерминиран хаос". Опитът показва, че добре обмислените инструменти за изследване, помагат и в "творческата" част, пестейки време и дори подсказващи понякога интересни идеи. Когато са самостоятелно създадени, те могат да се променят с цел да отговарят на специфични въпроси, интересуващи изследователя точно в дадения момент. Някои конкретни възможни подобрения и разширения включват:

1. Предоставяне на алтернатива за пресмятаните на някои математически функции и константи с използване на паралелни пресмятания в основната библиотека (вероятно втори екземпляр с друго пространство на имената, за да се използват същите имена и сигнатури, по възможност). Реализация на някои липсващи асимптотики по индекс и гранични случаи за няколко специални функции.

2. Добавяне на допълнителни средства за пресмятане на определени интеграли, включващи случая на бавно затихващи осцилиращи подинтегрални функции. Евентуално добавяне на алтернативни методи на ниво завършен програмен инструмент (освен реализираните до сега две схеми).

3. Създаване на подходящ интерфейс за общия случай на система ОДУ, осигуряващ удобство при въвеждане на задачата, без да се жертва ефективността. Евентуално създаване на различни програмни инструменти за конкретни модели на задачите, отчитащи спецификата и в интерфейса (например, различни за хамилтонови системи и дисипативни системи; пресмятане на показателя на Ляпунов).

4. Добавяне на възможности за пресмятане на още различни константи с голяма точност. Тук перспективите са много широки. За някои интересни случаи тепърва се изисква намиране на достатъчно ефективни методи.

5. Автоматизация при идентификация на константи.

Заключение - основни резултати.

Настоящият дисертационен труд е посветен на създаването на средства за разработка на програмни инструменти, работещи с произволна точност в конкретна програмна среда - Net Framework, а също реализацията на програмни инструменти в областта на числения анализ, използващи създадените за целта средства.

Основните научно-приложни резултати в настоящия дисертационен труд се свеждат до следното:

1. Осъществена е програмната реализация в целевата програмна среда за пресмятане на елементарни и специални математически функции, използвани при разработването на средствата за решаване на някои задачи на числения анализ. На базата на предложената реализация е разработено самостоятелен програмен инструмент SFCALC, даващ следните предимства: 1) облекчава възможността за тестване и 2) позволява удобно интерактивно генериране на начални условия, изисквани от други програмни инструменти в разработената система.

2. Изследвани са две от най-перспективните квадратурни схеми, предлагащи възможност за ефективно пресмятане на определени интеграли с висока точност. Предложена е методика на реализация, отчитаща особеностите на използваната среда на разработка, включително за паралелни пресмятания. Разработени са програмни инструменти за пресмятане на определени интеграли с висока точност. Един от тях (NQTS) демонстрира възможностите за създаване на интерактивно графично приложение в използваната среда. За другите два (THSHPar и CCPar) са предложени схеми на реализация, позволяващи паралелни пресмятания. За тази реализация са направени тестове, показващи възможността за ефективни пресмятания с много висока точност със специфичните средства на използваната среда, включваща масово вгражданите в съвременните преносими и настолни компютри многоядрени процесори.

3. Разработени са програмни инструменти за пресмятане на системи от обикновени диференциални уравнения (ОДУ) с много висока точност. Един от тях е специализиран (VODE2) и демонстрира възможностите за създаване на интерактивен графичен програмен инструмент в използваната среда за важен частен случай - уравнение от втори ред. Два други (IRK_Test, ODEX) дават възможност за третиране на обща задача за решаване на система ОДУ. Първият от тях, който е и основен за предлаганата система, дава някои предимства при по-продължителни симулации за някои класове задачи. Вторият, допълнителен, е с акцент върху ефективността.

4. Разработени са няколко програмни инструмента за пресмятане на скаларни нелинейни уравнения с много висока точност. Те обхващат реализация на бързо сходящи локални методи за случая, когато е налично достатъчно добро начално приближение - MPRootFindTestLocal, а също и реализация на глобален метод - в MPRootFindTestHN. Предвид тяхната практическа важност е осъществена и самостоятелна реализация за важни частни случаи - корени на специални функции, където видът на уравнението дава допълнителна информация. В повечето случаи тази реализация е вградена на ниво подпрограми непосредствено в основната библиотека. Освен това са изградени и самостоятелни програмни инструменти: за намиране на корените на ортогонални полиноми zrortpol; за намиране на корените на функции на Бесел от първи и втори вид zrbessel; за намиране на корените на функциите на Ейри и техните производни - aizeros, bizeros, ai_der_zeros, bi_der_zeros.

5. Изследвани са възможностите за бързо пресмятане на някои математически константи с висока точност. Предложените методи за реализация са разнообразни, в зависимост от пресмятаната константа. В някои случаи е възможно прилагането на изключително ефективното двоично разделяне (binary splitting), в други случаи се предлагат методи, специално адаптирани за пресмятане на конкретна константа. В редица случаи предложените средства използват възможностите за паралелни пресмятания, осигурявани от целевата среда. Всичко това е интегрирано в отделно създаден за целта програмен инструмент MPConst.

6. Разработен е метод за ефективна програмна реализация на алгоритъма PSLQ в неговия основен вариант за настоящата програмна среда. Работоспособността е проверена за нетривиални примери.

7. Разработени са допълнителни средства, свързани с облекчаване на въвеждането на задачите, между които нестандартен метод за символно диференциране.

8. Извършени са редица числени експерименти, които показват работоспособността на предложените в дисертационния труд методи и алгоритми, както и възможностите за тяхната практическа реализация.

Декларация за оригиналност на резултатите.

Декларирам, че настоящата дисертация съдържа оригинални резултати, получени при проведени от мен научни изследвания (с подкрепата и съдействието на научния ми консултант). Резултатите, които са получени, описани и/или публикувани от други учени, са надлежно и подробно цитирани в библиографията. Настоящата дисертация не е прилагана за придобиване на научна степен в друго висше училище, университет или научен институт.

Подпис:

Величко Г. Джамбов

Библиография

[1] 10000 digits of Khinchin's constant, computed in 11 minutes using mpmath+gmpy, http://mpmath.googlecode.com/svn/data/khinchin.txt

[2] Abramowitz, M., I. A. Stegun (Eds.), "Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables". National Bureau of Standards and Applied Mathematics Series. Vol. 55, 1964.

[3] Agafonov, E., "Multithreading in C# 5.0 Cookbook" , Packt Publishing, 2013.

[4] Allgower E., Georg, K., "Numerical Continuation Methods", Springer-Verlag, 1990.

[5] Almkvist, G., C. Krattenthaler, J. Petersson, “Some new series for π”, Experiment. Math. 12 (2003), 441–456.

[6] Amdeberhan, T., Faster and faster series for $\zeta(3)$, The Electronic Journal of Combinatorics, 3, 1996, http://arxiv.org/pdf/math/9804126.pdf

[7] Amdeberhan, T., D. Zeilberger, Hypergeometric series acceleration via the WZ method, Elec. J. Combin., 4, R3, 1997.

[8] Andrews, G. E., R. Askey, R. Roy, "Special Functions", Cambridge University Press, 1999.

[9] Artin, E., "The Gamma Function", Holt, Rinehart and Winston, 1964.

[10] Bailey, D., “A collection of mathematical formulas involving π”, manuscript, Nov 2010, http://www.davidhbailey.com/dhbpapers/pi-formulas.pdf.

[11] Bailey, D. H., "Experimental Mathematics and Optimization", invited short course presentation, McMaster University, Canada, Aug 2007.

[12] Bailey, D. H., "Peter Borwein and High-Performance Computing", 2008, http://crd-legacy.lbl.gov/~dhbailey/dhbtalks/dhb-peter-borwein.pdf.

[13] Bailey, D., “Pi Directory”, http://www.davidhbailey.com/pi/.

[14] Bailey, D. H., "The PSLQ Algorithm: Techniques for Efficient Computation", University of Newcastle (CARMA center), Newcastle, Australia, 24 Aug 2010, http://www.davidhbailey.com/dhbtalks/dhb-carma-20100824.pdf.

[15] Bailey, D., “Tanh-Sinh High-Precision Quadrature”, 2006,

http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/dhb-tanh-sinh.pdf.

[16] Bailey, D. H., Borwein, J. M. (2005). "Experimental mathematics: Examples, methods and implications." Notices of the American Mathematical Society, 52(5):502-514.

[17] Bailey, D., J. Borwein, “Hand-on-hand combat with thousand-digit integrals”, Journal of Computational Science, Volume 3, Issue 3, May 2012, 77-86.

[18] Bailey, D., J. Borwen, “Highly Paralell, Highly-Precision Numerical Integration”, 2008, http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/quadparallel.pdf.

[19] Bailey, D, H., J. M. Borwein, N. J. Calkin, R. Girgensohn, D. Russell Luke, V. H. Moll, “Experimental Mathematics in Action”, A K Peters Ltd., Wellesley, MA, 2007.

[20] Bayley, D., J. Borwein, R. Crandall, “On the Khintchine Constant”, Mathematics of Computation, vol. 66, 1997, 417–431.

[21] Bailey, D. H., Broadhurst, D. J., "Parallel Integer Relation Detection: Techniques and Applications", Math. Comp. 70 (2001), 1719-1736.

[22] Bailey, D.., D. Broadhurst, “Parallell Integer Relation Detection: Techniques and Applications”, Mathematics in Computation, vol. 70, no. 236, 2000, 1719-1736.

[23] Bailey, D. H., Broadhurst, D., Y., Li, X. S., Thompson, B., "High Performance Computing Meets Experimental Mathematics", Supercomputing, ACM/IEEE 2002 Conference, July 29, 2002.

[24] Bailey, D., K. Jeyabalan, X. Li, “A Comparison of Three High-Precision Quadrature Schemas”,” Experimental Mathematics, vol. 3, 2005, 317–329.

[25] Bateman, H., A. Erdélyi, "Higher Transcendental Functions" Vol. 1, 2, 3, McGraw-Hill Book Company, 1953-1955.

[26] Blewett, R., Clymer, A., "Pro Asynchronous Programming with .NET", APRESS, 2013.

[27] Borwein, P., "An Efficient Algorithm for the Riemann Zeta Function", Proc. of Canadian Mathematical Society Conference, 1991.

[28] Borwein, J. M., D. H. Bailey, “Mathematics by Experiment: Plausible Reasoning in the 21st Century”, 2nd Edition, A K Peters, Wellesley, 2008.

[29] Borwein, J., D. Bailey, R. Girgensohn, “Experimentation in Mathematics, Computational Paths to Discivery”, A K Peters Ltd., 2004.

[30] Borwein, J. M., P. B. Borwein, "Pi and the AGM: A Study in Analytic Number Theoryand Computational Complexity", A Wiley-Interscience Publication, JOHN WILEY & SONS, 1987.

[31] Borwein, J., P. Borwein, D. Bailey, “Ramanujan, Modular Equations, and Approximations to Pi or How to Compute One Billion Digits of Pi”, The American Mathematical Monthly, Vol. 96, No. 3, (Mar., 1989), 201-219.

[32] Borwein, J., Borwein, P., Girgensohn, R., Parnes, S. . "Making sense of experimental mathematics", The Mathematical Intelligencer, 18(4): 1996, 12-18.

[33] Brent, R. P, "Fast Multi-Precision Evaluation of Elementary Functions", Journal of the Associations for Computing Machinery, Vol. 23, April 1976, No 2, 242-251.

[34] Brent, R. P., "Fast Algorithms for High-Precision Computation of Elementary Functions", Presented at RNC7, Nancy, 12 July 2006, http://maths-people.anu.edu.au/~brent/pd/RNC7t.pdf.

[35] Brent, R., "The complexity of multi-precision arithemetic" в Anderssen,, R. S., Brent, R. P. (Eds) "The Complexity of Computational Problem Solving", Univ. of Queensland Press, 1976, 126-165.

[36] Brent, R. P, "Modern Computer Arithmetic", Cambridge University Press, 2011.

[37] Brent, R. P., . E . M. McMillan, "Some New Algorithms for High-Precision Computation of Euler’s Constant", Mathematics of Computation, Vol. 34, January 1980, No 149, 305-312.

[38] Brin, M., G. Stuck, "Introduction to Dynamical Systems", Cambridge University Press, 2003.

[39] Broer, H., F. Takens, "Dynamical Systems and Chaos", Springer, 2011.

[40] Butcher, J. C. "Numerical Methods for Ordinary Differential Equations", Second Ed. John Wiley & Sons, Ltd., 2008.

[41] Byrd, P. F., M. D. Friedman, "Handbook of Elliptic Integrals for Engineers and Scientists" Second Edition, Revised, Springer-Verlag, 1971.

[42] Campbell, C., R. Johnson, A. Miller, S. Toub, "Parallel Programming with Microsoft .NET. Design Patterns for Decomposition and Coordination on Multicore Architectures", Microsoft Series: patterns & practices, 2010.

[43] Chen, M.-P., H. M. Srivastava, Some familes of series representations for the Riemann ζ(3), Resultate Math. 33, 1998, 179–197.

[44] Cheng, H., B. Gergel, E. Kim, E. Zima, "Space-Efficient Evaluation of Hypergeometric Series", ACM SIGSAM Bulletin, vol. 39, No. 2 (2004), 41-52.

[45] Chudnovsky, D. V., Chudnovsky, G. V., "The Computation of Classical Constants", Proceedings of the National Academy of Sciences of the United States of America 86 (21),1989, 8178–8182.

[46] Clarckson, P. A., "Numerics and Asymptotics for the Painlevé Equations", in "Numerical solution of the Painlevé equations", ICMS, May 2010.

[47] Clarckson, P. A., "Painlevé Equations—Nonlinear Special Functions", in "Special Functions in the 21st Century", Washington DC, April 2011.

[48] Decarolis, F., R. Mayer, M. Santamaria, "An Algorithm for the Fixed Point and Newton Homotopy Methods with Some Examples", http://people.bu.edu/fdc/H-topy.pdf.

[49] Dekker, K., J. G. Verwer. "Stability of Runge–Kutta Methods for Stiff Nonlinear Differenrial Equations", North-Holland, 1984.

[50*] Djambov, V., "Using Implicit Runge-Kutta Methods for Multi Precision Solving of Nonlinear Differential Equations" Problems of Engineering, Cybernetics and Robotics, 62, 2010, 24-42.

[51] Dongarra, J., Sullivan F., "Guest Editors' Introduction: The Top 10 Algorithms", Computing in Science and Engineering Volume 2, Number 1, January/February, 2000, 22-23.

[52*] Dzhambov, V., S. Drangajov, "Computing of Special Functions with Arbitrary Precision in the Environment of .NET Framework", Cybernetics and Information Technologies, Volume 11, No 2, 2011, 32-45.

[53*] Dzhambov, V., "High Precision Computing of Definite Integrals with .NET Framework C# and X-MPIR", Cybernetics and Information Technologies, Volume 14, No 1, 2014, 172-182.

[54*] Dzhambov, V., "Solving Scalar Nonlinear Equations: High Precision Computation with .NET Framework C# and X-MPIR", Proceedings, International Workshop on "Advanced Control and Optimization: Step Ahead'2014", Prof. Marin Drinov Academic Publishing House, 2014, 11-17.

[55] Eckmann, J.-P., D. Ruelle, "Ergodic theory of chaos and strange attractors", Reviews of Modern Physics, Vol. 57, No. 3, Part I, July 1985, 617-656.

[56] Edwards, H. M., "Riemann's Zeta Function", Academic Press, 1974.

[57] Fabijonas, B., R., F. W. J. Olver, "On the Reversion of an Asymptotic Expansion and the Zeros of the Airy Functions", SIAM Rev. 41(4), 762-773.

[58] Ferguson, H. "PSOS: A New Integral Relation Finding Algorithm Involving Partial Sums of Squares and No Square Roots." Abs. Papers Presented to Amer. Math. Soc. 9, No. 56 88T-11-75, 214, Mar. 1988.

[59] Ferguson, H. R. P. , Bailey, D. H. "A Polynomial Time, Numerically Stable Integer Relation Algorithm." RNR Techn. Rept. RNR-91-032, Jul. 14, 1992.

[60] Ferguson, H. R. P., Bailey, D. H., Arno, S., "Analysis of PSLQ, An Integer Relation Finding Algorithm." Math. Comput. 68, 1999, 351-369.

[61] Ferguson, H. R. P., Forcade, R. W., "Generalization of the Euclidean Algorithm for Real Numbers to All Dimensions Higher than Two." Bull. Amer. Math. Soc. 1, 1979, 912-914.

[62] Finch, S. , "Mathematical constants", Cambridge University Press, 2003.

[63] Flajolet, P., Vardi, I., "Zata Function Expansions of Classical Constants", 1996, http://algo.inria.fr/flajolet/Publications/FlVa96.pdf.

[64] Freeman, A., "Pro .NET 4 Parallel Programming in C# 5.0", APRESS, 2010.

[65] Freeman, B., ".NET 4.5 Parallel Extensions Cookbook", Packt Publishing, 2013.

[66] Fortran and Matlab Codes, http://www.unige.ch/~hairer/software.html.

[67] Gauss, C. F., Werke (Gottingen, 1866-1933), vol. 3, pp. 361-403.

[68] Gil, A., Segura, J., "Computing the Zeros and Turning Points of Solutions of Second Order Homogeneous Linear ODES", SIAM J. Numer. Anal., Vol. 41 No. 3, 827-855.

[69] Gil, A., J. Segura, N. M. Temme в Simos, T. E. (Ed.), "Basic Methods for Computing Special Functions", "Recent Advances in Computational and Applied Mathematics", European Academy of Sciences, Springer, 2011.

[70] Gil, A., J. Segura, N. M. Temme. "Numerical Methods for Special Functions" SIAM, 2007.

[71] Glaser, A., Liu, X., Rokhlin, V., "A Fast Algorithm for the Calculation of the Roots of Special Functions", SIAM Journal on Scientific Computing 29(4), 2007, 1420-1438.

[72] Gosper, R. Wm., непбубликувано изследване, 1974, цитирано в Almkvist, G., C. Krattenthaler, J. Petersson, "Some new series for π", Experiment. Math. 12, 2003, 441–456.

[73] Gourdon, X., https://github.com/jlapeyre/mext/blob/master/packages/discrete_aex/khintchine.txt

[74] Haible, B., Papanikolaou, T., "Fast multiprecision evaluation of series of rational numbers", Algorithmic Number Theory, Lecture Notes in Computer Science Vol. 1423, 1998, 338-350.

[75] Hairer, E., C. Lubich, G. Wanner, "Geometric Numerical Integration, Structure-Preserving Algorithms for Ordinary Differential Equations", Second Edition, Springer, 2006.

[76] Hairer, E., S.P. Nørsett, G. Wanner, "Solving Ordinary Differential Equations I, Nonstiff Problems", Second Revised Edition, Springer, 1993.

[77] Harvey, D., "A multimodular algorithm for computing Bernoulli numbers", Mathematics of Computation, Vol. 79, Number 72, October 2010, 2361–2370.

[78] Harvey, D., "Old and new algorithms for computing Bernoulli numbers", 2012, http://web.maths.unsw.edu.au/~davidharvey/talks/bernoulli.pdf.

[79] Hastad, J., Just, B., Lagarias, J. C., Schnorr, C. P., "Polynomial Time Algorithms for Finding Integer Relations Among Real Numbers." SIAM J. Comput. 18, 1988, 859-881.

[80] Jost, J., "Dynamical Systems. Examples of Complex Behaviour", Springer, 2005.

[81] Judd K., "Numerical Methods in Economics", The MIT Press, Cambridge, Massachusetts, London England, 1998.

[82] Karatsuba, A. A., S. M. Voronin, "The Riemann Zeta-Function", Walter and Gruyter, 1992.

[83] Katok, A., B. Hasselblatt, "Introduction to the Modern Theory of Dynamical Systems", Cambridge University Press, 1997.

[84] Kelley, J. L., "General Topology", Van Nostrand, Princeton, 1955.

[85] Khintchine, A., "Continued Fractions", Univ. of Chicago Press, 3rd ed., 1961.

[86] Krtolica, P., Stanimirovi'c, P., "Symbolic Derivation without Using Expression Tree", Yugoslav Journal of Operations Research 11 (2001) Number 1. 61-75.

[87] Lenstra, A. K., Lenstra, H. W., Lovasz, L., "Factoring Polynomials with Rational Coefficients." Math. Ann. 261, 1982, 515-534.

[88] Lima, F., “A rapidly converging Ramanujan-type series for Catalan’s constant”, preprint, 2012, http://arxiv.org/abs/1207.3139.

[89] Lima, F. M. S., "An Euler-type formula for β(2n) and closed-form expressions for a class of zeta series", Integral Transforms and Special Functions Volume 23, Issue 9, 2012.

[90] Lind, D., B. Marcus, "An Introduction to Symbolic Dynamics and Codding", Cambridge University Press, 1999.

[91] Long, K., McKale, K. D., "New Tricks for an Old Dog: Archimedes, Gauss, and the Fast Computation of Stochastic Inverse Transcendentals", DOE Applied Math Meeting Oct 2011, http://www.csm.ornl.gov/workshops/applmath11/documents/posters/Long_poster.pdf.

[92] Lupas, A., Formulae for some classical constants. In Proceedings of ROGER-2000, 2000, http://www.lacim.uqam.ca/~plouffe/articles/alupas1.pdf.

[93] Martelli, M., M. Dang, T. Seph, "Defining Chaos", Mathematics Magazine, Vol. 71, No 2, April 1998, 112–122.

[94] McGown, K. J., "Computing Bernoulli Numbers Quickly", December 8, 2005, http://wstein.org/projects/168/kevin_mcgown/bernproj.pdf.

[95] McNamee J., "Numerical Methods for Roots of Polynomials - Part I", Elsevier, 2007.

[96] McNamee J.M., Pan V.Y., "Numerical Methods for Roots of Polynomials - Part II", Elsevier, 2013.

[97] Miel, G., "Of calculations past and present: the Archimedean algorithm", American Mathematical Monthly, 90, 1983, pp. 17-35.

[98] Milnor, J., "On the Concept of Attractor", Communications in Mathematical Physics, 99, 1985, 177-195.

[99] Milnor, J., "On the Concept of Attractor: Correction and Remarks", Communications in Mathematical Physics, 102, 1985, 517-519.

[100] Mori, M., "Discovery of Double Exponential Transformation and its Developments", Publications of RIMS, Kyoto University, vol. 41, 2004, 897-935.

[101] MPIR site, http://www.mpir.org/.

[102] Muller, J.-M., "Elementary Functions. Algorithms and Implementation.", Birkhäuser, 2006.

[103] NIST Digital Library of Mathematical Functions (Site), http://dlmf.nist.gov/

[104] Numbers, constants and computation (site, Xavier Gourdon and Pascal Sebah), http://numbers.computation.free.fr/Constants/constants.html.

[105] Olver, F. W. J., "Asymptotics and Special Functions", Academic Press, 1974.

[106] Olver, F. W. J., D. W. Lozier, R. F. Boisvert, C. W. Clark (Eds.) "NIST Handbook of Mathematical Functions", National Instutute of Standards and Technology, Cambridge University Press, 2010.

[107] Ooura, T., M. Mori, "The double exponential formula for oscillatory functions over half infinite interval", J. Comp. Apl. Math. 38, 1991, 353-360.

[108] Ooura, T., "A Double Exponential Formula for Fourier Transforms", Publications of RIMS, Kyoto University, vol. 41, 2005, 971-977.

[109] Ostrowski A., "Solution of Equations and Systems of Equations", Academic Press, 2nd edition, 1966.

[110] Peetree, J., "GENERALIZING THE ARITHMETIC GEOMETRIC MEAN A HAPLESS COMPUTER EXPERIMENT", Intenat. J. Math. & Math. Sci. VOL. 12 NO. 2, 1989, 235-246.

[111] PiFast, http://numbers.computation.free.fr/Constants/PiProgram/pifast.html, 2004.

[112] Rahimian S., Jalali, F., White, R., "A New Homotopy for Seeking All Real Roots of a Nonlinear Equation", Computers and Chemical Engineering 35 (2011) 303-411.

[113] Ramanujan, S., "Modular equations and approximations to Pi." Quart. J. Pure. Appl. Math. 45, 1913-1914, 350–372.

[114] Ramanujan, S., On the integral $\int_0^x \frac{\tan^{-1} t}{t} dt$, J. Indian Math. Soc. VII, 93-96, 1915.

[115] Ringler, R., "C# Multithreaded and Parallel Programming", Packt Publishing, 2014.

[116] Robinson, C., "Dynamical Systems. Stability, Symbolic Dynamics, and Chaos", CRC Press, 1995.

[117] Robinson, C., "What is chaotic attractor?", Qualitative theory of Dynamical Systems, Vol. 7, No. 1, 2008, 227-236.

[118] Rovelli, C., "Forget time", Essay written for the FQXi contest on the Nature of Time, August 24, 2008, http://arxiv.org/pdf/0903.3832.pdf.

[119] Salamin, E. "Computation of pi Using Arithmetic-Geometric Mean." Math. Comput. 30, 565-570, 1976

[120] Sarra, S. A., C. Meador, "On the numerical solution of chaotic dynamical systems using extend precision floating point arithmetic and very high order numerical methods", Nonlinear Analysis: Modelling and Control, 2011, Vol. 16, No. 3, 340–352.

[121] Sasaki, T., Kanada, Y. "Practically fast multiple-precision evaluation of log(x)". Journal of Information Processing 5 (4), 1982, 247–250.

[122] Segura, J., "The zeros of special function from a fixed point method", SIAM J. Numer. Anal., 40 (2002), 114-133.

[123] Shanks, D., "The second-order term in the asymptotic expansion of B(x)", Math. Comp. 18 (1964) 75–86.

[124] Sharma R., "Iterative Methods for the Solution of Nonlinear Equations", A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Mathematics, Department of Mathematics Sant Longowal Institute of Engineering and Technology Longowal"- 148 106, District Sangrur, Punjab (India), 2011.

[125] Smullyan, R. M., "Theory of Formal Systems", Princeton Unuversity Press, 1962.

[126] Solovay, R. M., "A model of set theoty in wich every set of reals is Lebesgue measurable", Analls of Math., 92 (1970), 1-56.

[127] Sørensen, H. K., "Exploratory experimentation in experimental mathematics: A glimpse at the PSLQ algorithm", в Benedikt Löwe & Thomas Müller (eds.), PhiMSAMP. Philosophy of Mathematics: Sociological Aspects and Mathematical Practice. College Publications, 2010, 341-360.

[128] Spouge, J. L., "Computation of the Gamma, Digamma and Trigamma Functions", SIAM Journal on Numerical Analysis, 31 , 1994, No 3, 931-944.

[129] Sprott, J. C., "Elegant Chaos. Algebraically Simple Chaotic Flows", World Scientific, 2010.

[130] Srivastava, H. M., Choi, J., "Zeta and q-Zeta Functions and Associated Series and Integrals", Elsevier, 2012.

[131] Staudt, K. G. C. von. "Beweis eines Lehrsatzes, die Bernoullischen Zahlen betreffend", J. Reine Angew. Math., 21, 1840, 372-374.

[132] Sun, Z.-W., "Open conjectures on congruences", arXiv:math.NT/0911.5665v59, 2011.

[133] Sun, Z.-W., "Fast converging series for some famous constants", preprint, 2010, http://arxiv.org/abs/1010.4298.

[134] Szegö G., "Orthogonal Polynomials", American Mathematical Society, 1939.

[135] Takahasi, H., M. Mori, “Double Exponential Formulas for Numerical Intergration”, Publications of RIMS, Kyoto University, vol. 9, 1974, 721-741.

[136] Tanaka, K., M. Sugihara, K. Murota, M. Mori, “Function Classes for Double Exponential Integration Formulas”, Methematical Engineering Technical Reports, Tokyo University, 2007.

[137] Titchmarsh, E. C., "The Theory of the Riemann Zeta-Function", Second Edition, Reprinted, Oxford, Clarendon Press, 1988.

[138] Traub J., "Iterative Methods for the solution of Equations", Prentice-Hall. London, 1964.

[139] Volpi L., "Non Linear Equations: Practical Methods for Root Finding, Part II - Zeros of Polynomials", Foxes Team, 2006.

[140] Watson, G. N., "A Treatise on the Theory of Bessel Functions" Second Edition. Cambridge University Press, 1945.

[141] Wedeniwski, S.,, The Value of Zeta(3) to 1,000,000 places, 1998, http://www.math.rutgers.edu/~zeilberg/mamarim/mamarimhtml/Zeta3.txt

[142] Wiggins, S., "Introduction to Nonlinear Dynamical Systems and Chaos", Second Edition, Springer, 2005.

[143] Wikipedia site, http://en.wikipedia.org/

[144] Wolfram Mathworld site, http://mathworld.wolfram.com/.

[145] Wolfram Functions site, http://functions.wolfram.com/.

[146] XMPIR, http://www.alglib.net/x/xmpir/xmpir-0.2.zip.

[147] Ye, L. , “Numerical quadrature: Theory and computation”, Submitted in partial fulfillment of the requirements for the degree of Master of Computer Science at Dalhousie University Halifax, Nova Scotia August 2006, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.152.8467&rep=rep1&type=pdf

[148] Yee, A., http://www.numberworld.org/y-cruncher/#Download, 2015.

[149] Yee, A., S. Kondo, "10 Trillion Digits of Pi: A Case Study of summing Hypergeometric Series to high precision on Multicore Systems", preprint, 2011, http://hdl.handle.net/2142/28348.

[150] Александров П. С. "Введение в теорию множеств и общую топологию", Наука, 1977.

[151] Успенский, В. А., "Лекции о вычислимых функциях", Государственное издательство физико-математиескиой литературы, Москва, 1960.

[152] Armitage J. V.,Eberlein W. F., Elliptic Functions, Cambridge University Press 2006

[153] Bailey D. H., Barrio R., Borwein J. M., "High-Precision Computations: Mathematical Physics and Dynamics", Applied Mathematics and Computation, Volume 218, Issue 20, 15 June 2012, pp. 10106-10121

[154] http://mpir.org/

[155] Clenshaw C. W. and Curtis A. R., "A method for numerical integration on an automatic computer", Numerische Mathematik 2, 197-205 (1960)

[156] Boyd J. P., "Chebyshev and Fourier Spectral Methods", Second Edition (Revised), Dover Publications, 2001

[157] Mason J. C. ,Handscomb D. C., "Chebyshev polynomials", CHAPMAN & HALL/CRC, 2003

[158] Trefethen L. N., "Six Myths of Polynomial Interpolation and Quadrature", https://people.maths.ox.ac.uk/trefethen/mythspaper.pdf

[159] Trefethen L. N., "Is Gauss Quadrature Better than Clenshaw–Curtis?", SIAM Review, Volume 50 Issue 1, Pages 67-87, February 2008.

[160] Kang Feng, Mengzhao Qin, Symplectic Geometric Algorithms for Hamiltonian Systems, Springer, 2010.

[161] Arndt J., "Matters Computational", Springer, 2011.

[162] Waldvogel J., "Fast Construction of the Fejér and Clenshaw–Curtis Quadrature Rules", BIT Numerical Mathematics, March 2006, Volume 46, Issue 1, pp 195-202.

[163] Kythe, P., K. Michel, R. Schäferkotter. Handbook of Computational Methods for Integration. Chapman & Hall/CRC Press, 2005.

[164] Zehnder, E., "Lectures on Dynamical Systems", European Mathematical Society Publishing House, 2010.

[165] Sprott, J. C., "Elegamt Chaos. Algebraically Simple Chaotic Flows", World Scientific, 2010.

[166] Malasoma J.-M., "A new class of minimal chaotic flows", Phys. Lett. A 305, pp. 52-58,2002.

[167] Moore D. W., Spiegel E. A., "A thermally excited non-linear oscillator", Astrophys. J. 143, pp. 871-887, 1966.

[168] Thomas R., "Deterministic chaos seen in terms of feedback circuits: Analysis, Synthesis, 'labyrinth chaos'", Int. J. Bifurcat. Chaos Appl. Sci., pp. 1889-1905, 1999.

[169] Chua L. O., "Chua's circuits. Ten years later", IEICE Trans. Fund. Electron. Comm. Comput. Sci. E77-A pp. 1811-1822, 1994.

[170] Hoshi M., Kono M., "Rikitake two-disk dynamo system: Statistical properties and growth of instability", J. Geophys. Res. 93, pp. 11643-11654, 1988.

[171] Robey R. W., Robey J. M. and Aulwes R., "In search of numerical consistency in parallel programming," Parallel Computing, vol. 37 (2011), 217-219.

[172] Bailey D. H., Li X. S. and Thompson B., "ARPREC: An arbitrary precision computation package" Sep 2002, http://crd.lbl.gov/~dhbailey/dhbpapers/arprec.pdf.

[173] Meschkowski H., Nilson W. (Eds.), " Georg Cantor Briefe" Sep 2002, Springer, (1991)

Списък на публикациите по дисертационния труд:

1. Djambov, V., "Using Implicit Runge-Kutta Methods for Multi Precision Solving of Nonlinear Differential Equations", Problems of Engineering, Cybernetics and Robotics, Volume 62, 2010, pp. 24-42, ISSN (Print) 0204-9848, ISSN (Online) 1314-409X.

2. Dzhambov, V., S. Drangajov, "Computing of Special Functions with Arbitrary Precision in the Environment of .NET Framework", Cybernetics and Information Technologies, Volume 11, No 2, 2011, pp. 32-45, ISSN (Print) 1311-9702.

3. Dzhambov, V., "High Precision Computing of Definite Integrals with .NET Framework C# and X-MPIR", Cybernetics and Information Technologies, Volume 14, No 1, 2014, pp. 172-182, ISSN (Print) 1311-9702, ISSN (Online) 1314-4081.

4. Dzhambov, V., "Solving Scalar Nonlinear Equations: High Precision Computation with .NET Framework C# and X-MPIR", Proceedings, International Workshop on "Advanced Control and Optimization: Step Ahead'2014", Prof. Marin Drinov Academic Publishing House, ISSN 1314-4634, pp. 11-17, 2014.

5. Dzhambov, V. "Solving Nonlinear Ordinary Differential Equations in .NET Framework Environment with X-MPIR", Information Technologies and Control. Volume 12, Issue 2, Pages 2–8, ISSN (Online) 1312-2622, DOI: 10.1515/itc-2015-0010, December 2015.

6. Dzhambov, V., V. Sgurev, "On the fast computing of some constants", Comptes rendus de l'Académie bulgare des Sciences, Tome 69, No 1, 2016, pp. 11-18, ISSN (Print) 1310-1331, ISSN (Online) 2367-5535.