Abstracts of Dissertations

Institute of Information and Communication Technologies

BULGARIAN ACADEMY OF SCIENCES

1 / 2017

METHODS, WAYS, AND MEANS TO IMPROVE HIGH PRECISION COMPUTATION OF SOME CLASSES OF PROBLEMS

Velichko Dzhambov

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

Величко Джамбов

Автореферати на дисертации

Институт по информационни и комуникационни технологии

БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ

ISSN: 1314-6351

Поредицата ***„Автореферати на дисертации на Института по информационни и комуникационни технологии при Българската академия на науките“*** *представя в електронен формат автореферати на дисертации за получаване на научната степен „Доктор на науките” или на образователната и научната степен „Доктор”, защитени в Института по информационни и комуникационни технологии при Българската академия на науките. Представените трудове отразяват нови научни и научно-приложни приноси в редица области на информационните и комуникационните технологии като Компютърни мрежи и архитектури, Паралелни алгоритми, Научни пресмятания, Лингвистично моделиране, Математически методи за обработка на сензорна информация, Информационни технологии в сигурността, Технологии за управление и обработка на знания, Грид-технологии и приложения, Оптимизация и вземане на решения, Обработка на сигнали и разпознаване на образи, Интелигентни системи, Информационни процеси и системи, Вградени интелигентни технологии, Йерархични системи, Комуникационни системи и услуги и др.*

The series ***Abstracts of Dissertations of the Institute of Information and Communication Technologies at the Bulgarian Academy of Sciences*** *presents in an electronic format the abstracts of Doctor of Sciences and PhD dissertations defended in the Institute of Information and Communication Technologies at the Bulgarian Academy of Sciences. The studies provide new original results in such areas of Information and Communication Technologies as Computer Networks and Architectures, Parallel Algorithms, Scientific Computations, Linguistic Modelling, Mathematical Methods for Sensor Data Processing, Information Technologies for Security, Technologies for Knowledge management and processing, Grid Technologies and Applications, Optimization and Decision Making, Signal Processing and Pattern Recognition, Information Processing and Systems, Intelligent Systems, Embedded Intelligent Technologies, Hierarchical Systems, Communication Systems and Services, etc.*

e-ISSN: 1314-6351

BULGARIAN ACADEMY OF SCIENCES

Abstract of PhD Thesis

METHODS, WAYS, AND MEANS TO IMPROVE HIGH PRECISION COMPUTATION OF SOME CLASSES OF PROBLEMS

Velichko Georgiev Dzhambov

Supervisor: Acad. Vassil Sgurev

Approved by Supervising Committee:

Prof. Ivan Dimov
Acad. Vassil Sgurev
Acad. Ivan Popchev
Acad. Mincho Hadzhiyski
Acad. Peter Popivanov

INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGIES
Department of Intelligent Systems

The PhD thesis was discussed and allowed to be defended during an extended session of the Department of Intelligent systems at IICT-BAS, which had been held on October 25, 2016.

The defense of the PhD thesis had been held on .March 14, 2017 at 10:30 am in Room 507, Block 25A, IICT-BAS.

The full volume of the dissertation is 176 pages. It consists of an introduction and eight chapters. The list of references contains 173 titles (p. 167-175). The text of the dissertation includes 17 tables and 56 figures.

Keywords: computation, numerical analysis, high precision, arbitrary precision, .Net Framework, MPIR, X-MPIR

1. Dissertation's main objective

The present dissertation investigates possibilities to implement methods and means concerning effective high precision computation related to the solution of some classes of problems in a specific software environment - .Net Framework in Windows operating system. That is to create an instrumental software system assisting development, which uses suitably all capabilities of this environment.

The main objective is as follows:

> To give researchers and developers, having experience with the specific environment an opportunity to create their own software, using arbitrary precision computations which is especially tuned to solve problems of their interests.

This main goal includes

> Creating means concerning development of software tools, working with arbitrary precision in the specific software environment (.Net Framework), which can be integrated directly in the process of development.
>
> Implementation of specific software instruments, related to the numerical analysis, which use the especially created tools.
>
> Investigate and possibly implement the capabilities for parallel computations, provided by the multi-core architecture, inherent to the processors on a mass scale built in all contemporary desktop and laptop computers.

On the other hand the above objective require

> Development of methods that suit requirements, specific to the software system being created, namely - they must be effective enough for arbitrary precision computations.

2. Motivation

Almost all contemporary computers implement the standard IEEE with 64-bit floating point arithmetic which ensures 53-bit mantissa - that is precision of around 15-16 decimal digits precision. In most cases this precision is enough, but an important exception also exist [29]. Some cases in which high precision computations are useful:

- Ill conditioned linear systems: often encountered scenario in the process of solving some more general problem when the 64-bit precision causes generation of errors.

- Large quantity of summations: anomalous results originated by loss of associativity in summations, particularly when executed on a parallel computer environment [31].

- Long simulations: various kinds of physical simulations, performed over many time intervals that result to depart from reality, due to cumulative round-of error, in addition to errors arising from discretization of time and space.

- Large-scale simulations: Computations that are well-behaved on modest-sized problems, may exhibit significant numerical errors when scaled up massively parallel systems.

- Resolving small-scale phenomena: It is often necessary to employ a very fine-scale resolution to "zoom" in on the phenomena in question.

"Experimental mathematics" computations: Numerous recent results in experimental mathematics could not be obtained except by very high precision computations.

All this makes it necessary to create relevant tools, that allow to perform high precision computations, if possible of high level, to make it easier their applying in various programming environments - a very important reason to create such special tools targeting .NET Framework in Windows operating system. Formally Windows allows using already developed libraries supporting high precision computations. In practice however using such libraries requires simulating some kind of Unix-like environment. Furthermore virtually without exceptions the interface of high level offered include C/C++ and several other high level languages, but not C# - the main language of .NET Framework.

Even though this concrete environment is widespread for personal computers, it seems to be underestimated by the software developers for scientific applications. Of course, there are reasons for this, but in our opinion, there are advantages as well: Comparatively easy integration in various functionalities such as visualization and interactivity in one and the same application if the necessary methods for solving the concrete problems are available, which is the purpose of the main library being created. The implementation of some basic tools makes it possible for the interested researcher to create his own applications in the given environment, exactly matching his concrete interests, viz. create and not be a user of a ready system. In addition this allows integrating easily the very valuable .NET Framework's "infrastructure" at the stage of design and development, which covers various themes not included by any ready system. And of course such approach makes it possible to use researcher's experience and skills already accumulated in a known environment. This work ~~is~~ presents one possible version of creating such a programming system.

3. Methodology

3.1 Methods and means to improve high precision computations of elementary and special functions

The background necessary to achieve dissertation's end purpose includes means and methods to compute with high precision elementary and special functions (Chapter 1) as well as some auxiliary ways and means for development of software tools (described in the introductory chapter). For that purpose effective methods are developed, thus embedding in the software tools being built, related to the solution of some numerical analysis problems. Main sources here are [21], [1], [6], [9], [7], [5], [14], [20], [13], [26], [25], [19]. Using the methods, realized at this stage, special graphical programming application SFCALC is implemented. This application gives two additional advantages. One of these is a fast testing of results obtained during development of other programming applications. The second advantage is the possibility for interactive generation of initial data required by other programming applications developed to solve specific classes of problems.

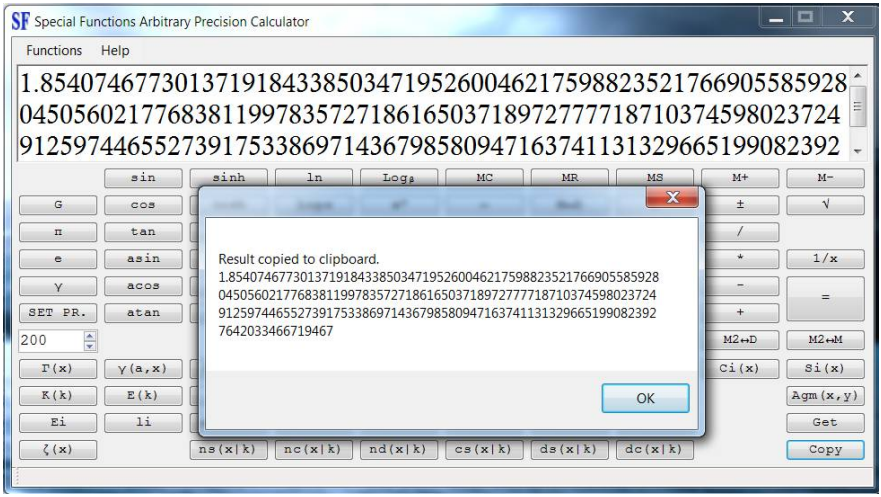


Fig. 1.4 Copying the result from SFCALC to the clipboard

The remaining part of the dissertation is devoted to a study of possibilities for solving specific classes of numerical analysis problems with arbitrary precision in the targeted software environment as well as practical software implementation. In addition an implementation of algorithm for integer relation detection is realized. Such type of algorithms is one of the main reasons to perform computations with arbitrary precision.

3.2 Methods to improve high precision numerical computation of some classes of definite integrals ~~classes~~

Two of most perspective quadrature schemas are studied (based on the so called Double Exponential Formulas [23] and on the Clenshaw-Curtis quadrature formula [30]), offering possibility for effective computation of definite integrals with very high precision. Methods are developed for ~~an~~ implementation rendering an account of the special features available in the targeted software environment, including parallel computations.

An implementation of special graphical programming application is realized in the proposed software system - NQTS, as well as two console programs (THSHPar, CCPar). In THSHPar and CCPar parallel computations are in use. NQTS and THSHPar use Double Exponential Formula. CCPar uses quadrature schema based on Clenshaw-Curtis.quadrature formula.

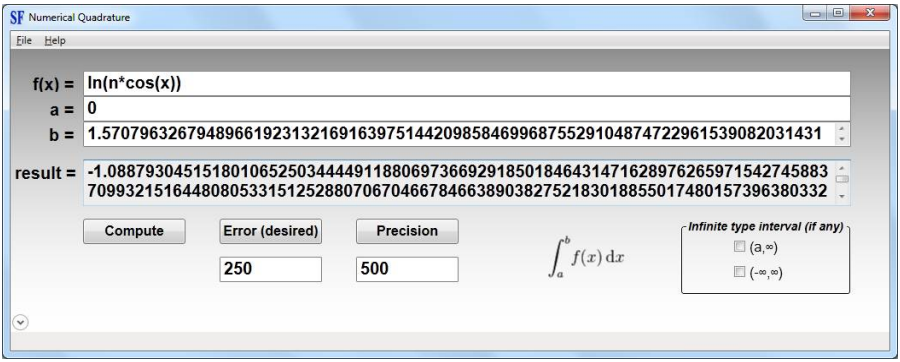


Fig. 2.8 Result from numerical computing of the integral $\int_0^{\frac{\pi}{2}} \ln(\cos(x))\,dx$ using NQTS.

Comparative tests are realized for THSHPar and CCPar. This comparison uses for base [3] where special hardware for parallel computations is applied.

3.3 Methods to improve the high precision numerical computations of systems of ordinary differential equations

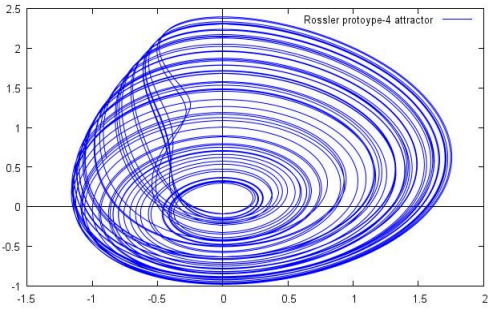
Possibilities for solving systems of ordinary differential equations with very high precision are investigated. The meaning and limitations concerning such type of solving are discussed. The specific choice of method for the main solver (IRK_Test) is implicit Runge-Kutta scheme of arbitrary order ([12], [10], [16]). This choice gives some advantages for solving specific classes of problems, namely conservative hamiltonian systems. Besides this main solver the proposed software system contains one more solver which is based on the Gragg-Burlish-Stoer extrapolation method, [17].

Rössler prototype - 4 attractor

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

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

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



Memory ciscillator

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

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

Halvorsen's circulant system

$$\begin{cases} \dot{x} = 1.3x - 4y - 4z - y^2 \\ \textit{plus cyclic exchange} \\ \left(x, y, z\right) \to \left(y, z, x\right) \to \left(z, x, y\right) \end{cases}$$

$$\left(x_0, y_0, z_0\right) = \left(-6.4, 0, 0\right)$$

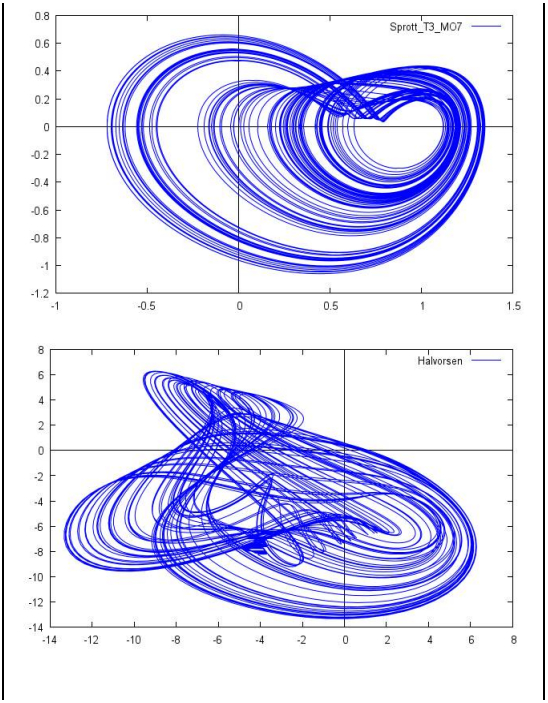


Fig. 3.2 Examples of solutions obtainable with IRK_Test

In addition an independent interactive software tool is developed, VODE2, especially designed for the case of solving nonlinear ordinary differential equation of second order.

This tool gives an additional visualization possibility in so called functional mode, which is useful sometimes - for example to examine the zero location of a function.

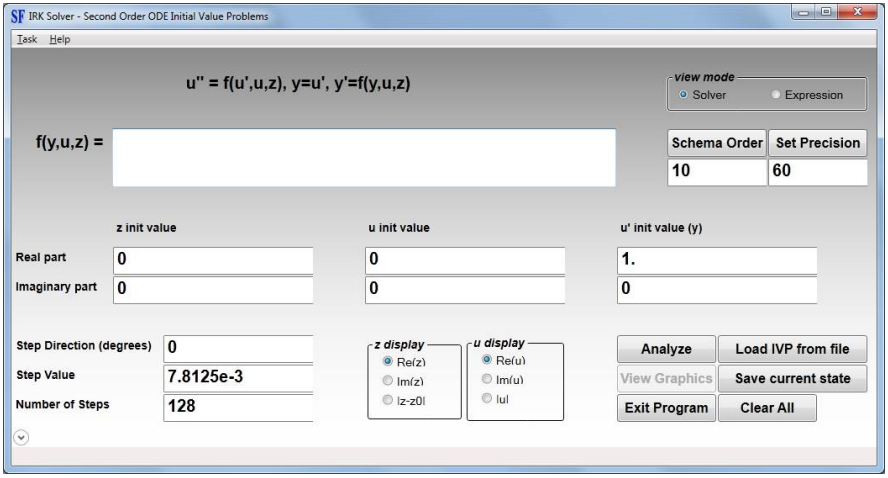


Fig. 3.3 The program VODE2 after starting

VODE2 allows parameters and can make use of SFCALC interactively to assign initial conditions.

3.4 Methods to improve root finding of scalar equations with high precision

Possibility for solving nonlinear equation with very high precision is investigated. Depending on the previously available information concerning the root and the function f in the equation f(x) = 0 various approaches are possible. When additional information is not available a global method is executed. In case of good enough information to localize the root fast converging iterative methods can be applied. If additional information for the function is available (for example, function that is solution of homogeneous linear differential equation of second order) applying special methods is possible reporting on this information. In the proposed software system all three possibilities are developed. The software tool in case of missing additional information is MPRootFindTestHN. It is an implementation of a variant of a Homotopy Continuation Method [2]. The software tool in case of good initial root localization is MPRootFindTestLocal - this one envelops several fast iterative methods ([24], [22]). The specific method to use is assigned to one of the input parameters. This software tool uses internally symbolic differentiation. The nonstandard realization of the symbolic differentiation proposed is described in the introductory chapter. For the zeros of some special functions, used in the main library, ~~an~~other particular methods are realized. Besides independent software tools are realized: zrortpol – for finding roots of orthogonal polynomials, zrbessel - for zeros' finding in case of Bessel

functions of first and second kind, as well as aizeros, bizeros, ai_der_zeros, bi_der_zeros for the zeros of Airy functions and their derivatives.

3.5 Fast computation of mathematical constants with high precision

Methods are investigated, that are related to fast computation with very high precision of some mathematical constants ([8], [15]). An implementation is proposed, that uses parallel computations for some frequently used mathematical constants Tests demonstrating effectiveness in comparison with leading contemporary program are performed ([27],[28]). Besides this several implementations of some more special constants are proposed.

There are at least two reasons that make mathematical constants subject of particular interest in high precision computing. Some of these like π, e, ,γ, ln 2, are frequently used in the process of computing transcendental mathematical functions (special and elementary). Additionally, high precision computing of mathematical constants is a preliminary condition if we want to implement some algorithm for integer relation detection (for example PSQL). The special application MPConsts proposed here envelops several methods devoted to effective high precision computing of some mathematical constants. Here is one of the places where parallel computations are used. (The other is related to the implementation of parallel realization of some quadrature schemes described in Chapter 2).

3.6 Integer relation detection and identification of constants. PSLQ algorithm

An effective software implementation of an algorithm for integer relation detection - PSLQ (Partial Sums using LQ decomposition) is proposed for the basic version of this algorithm (there exist another, multi-pair, also) [4], adapted to the targeted software environment.

Integer Relation Detection solve the following problem: given *n* real numbers $x_1, x_2, \ldots, x_n$, find if there exist *n* integer numbers (not all equal 0) $a_1, a_2, \ldots, a_n$, for which $a_1x_1 + a_2x_2 + \ldots + a_nx_n = 0$. This equation of course is accurate to the "epsilon" of the precision used.

Successful realization of ~~a~~ such an algorithm is a necessary step to implement software tools of type identify, present in some systems realizing elements of computer algebra.

3.7 Applications connected with the high precision computation of Systems of Ordinary Differential Equations

Results from practical solving systems of ordinary differential equations with very high precision are presented, exhibiting chaotic behavior - solutions obtained through applying especially proposed software tools of the presented software system. The first section contains illustrations related to the discussion in section 3.1, Chapter 3, concerning possibilities and limitations for quantitative study in the process of solving with gradually increasing precision. The second section contains examples illustrating the solutions of simple smooth dynamical systems, exhibiting chaotic behavior.

3.8 Applications connected with the fast high precision computation of some mathematical constants

A typical source code is presented, that allows applying of parallel computations via binary splitting. In addition representations through series of some frequently used mathematical constants related to convenient applying of binary splitting are collected.

Contribution summary

The contribution of the thesis can be summarized as follows:

1. Software implementation allowing arbitrary precision computation of elementary and special functions is realized, especially designed for the targeted software environment. This one is used in the process of solving some numerical analysis problems. Using this as base a separate software tool SFCALC is implemented, which gives the following advantages: 1) makes lighter the testing process and 2) allows handy interactive generation of initial conditions, required by other software tools in the developed software system.

2. Two of the most perspective quadrature schemes are studied, offering possibility for effective computation of definite integrals with very high precision. Methods are developed for an implementation rendering an account of the special features available in the targeted software environment, including parallel computations. Software tools for computing definite integrals with very high precisions are developed. One of these (NQTS) demonstrates the possibilities to create interactive graphical programming application in the software environment used.. For the two other (THSHPar и CCPar) schemes of realization are proposed that allow parallel computations. For this realization tests are performed, demonstrating the possibility of effective computations with very high precision, using ways and means specific to the targeted programming environment, which is

typical for many of the contemporary desktop and laptop computers that include multi-core processors.

3. Software tools for solving systems of ordinary differential equations with very high precision are developed. One is specialized (VODE2) and demonstrate the possibilities to create interactive graphical programming application in the targeted software environment for a special but important case - ordinary differential equation (ODE) of second order. Two others (IRK_Test, ODEX) allow solving in case of the general problem (system of ODEs in normal form). The first of these that is basic for the proposed software system has some advantages in case of long simulations and specific problem classes. The second, supplementary, is designed for speed.

4. Several software tools for solving nonlinear scalar equations with very high precision are developed. They encompass realization of fast converging local methods in case of good enough initial approximation - MPRootFindTestLocal, as well as global method - in MPRootFindTestHN. In view of their practical importance separate implementation for special cases is provided - zeros of special mathematical functions, where additional information is available (for example the function is solution of homogeneous ODE of second order). In most cases this implementation is embedded directly at subroutine level in the main library. In addition independent programming applications are built to find roots of classical orthogonal polynomials - zrortpol; finding zeros of Bessel functions of first and second kind - zrbessel; and finding zeros of Airy functions and their derivatives - aizeros, bizeros, ai_der_zeros, bi_der_zeros.

5. Possibilities for fast computing of some mathematical constants are studied. The methods of realization proposed are various depending on the constant to be computed. In some cases applying the very effective binary splitting is possible. Sometimes ~~an~~other methods are used, including special adaptation of known representations. In many cases the methods proposed use possibilities of parallel computations, offered by the targeted software environment. All this is integrated in a specially created for that purpose programming application MPConst.

6. An effective software implementation of the PSLQ algorithm is realized for the base version of this algorithm, adapted for the targeted software environment. The fitness is checked for non-trivial examples.

7. Additional auxiliary means are developed that allow user's input conveniently handling, including non-standard symbolic differentiation.

8. A series of numerical experiments are performed that manifest the effectiveness of methods and algorithms proposed as well as the possibility of their practical implementation.

References

[1] 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.

[2] Allgower E., Georg, K., "Numerical Continuation Methods", Springer-Verlag, 1990.

[3] Bailey, D., J. Borwen, “Highly Paralell, Highly-Precision Numerical Integration”, 2008, http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/quadparallel.pdf.

[4] 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.

[5] Bateman, H., A. Erdélyi, "Higher Transcendental Functions" Vol. 1, 2, 3, McGraw-Hill Book Company, 1953-1955.

[6] 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.

[7] 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.

[8] 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.

[9] Brent, R. P, "Modern Computer Arithmetic", Cambridge University Press, 2011.

[10] Butcher, J. C. "Numerical Methods for Ordinary Differential Equations", Second Ed. John Wiley & Sons, Ltd., 2008.

[11] 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.

[12] Dekker, K., J. G. Verwer. "Stability of Runge–Kutta Methods for Stiff Nonlinear Differenrial Equations", North-Holland, 1984.

[13] 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.

[14] Gil, A., J. Segura, N. M. Temme. "Numerical Methods for Special Functions" SIAM, 2007.

[15] 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.

[16] Hairer, E., C. Lubich, G. Wanner, "Geometric Numerical Integration, Structure- Preserving Algorithms for Ordinary Differential Equations", Second Edition, Springer, 2006.

[17] Hairer, E., S.P. Nørsett, G. Wanner, "Solving Ordinary Differential Equations I, Nonstiff Problems", Second Revised Edition, Springer, 1993.

[18] Judd K., "Numerical Methods in Economics", The MIT Press, Cambridge, Massachusetts, London England, 1998.

[19] Numbers, constants and computation (site, Xavier Gourdon and Pascal Sebah), http://numbers.computation.free.fr/Constants/constants.html.

[20] Olver, F. W. J., "Asymptotics and Special Functions", Academic Press, 1974.

[21] 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.

[22] 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.

[23] Takahasi, H., M. Mori, “Double Exponential Formulas for Numerical Intergration”, Publications of RIMS, Kyoto University, vol. 9, 1974, 721-741.

[24] Traub J., "Iterative Methods for the solution of Equations", Prentice-Hall. London, 1964.

[25] Wikipedia site, http://en.wikipedia.org/

[26] Wolfram Functions site, http://functions.wolfram.com/.

[27] Yee, A., http://www.numberworld.org/y-cruncher/#Download, 2015.

[28] 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.

[29] 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. 10106-1012

[30] Clenshaw C. W. and Curtis A. R., "A method for numerical integration on an automatic computer", Numerische Mathematik 2, 197-205 (1960)

[31] Robey R. W., Robey J. M. and Aulwes R., "In search of numerical consistency in parallel programming," Parallel Computing, vol. 37 (2011), 217-219.

БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ

АВТОРЕФЕРАТ НА ДИСЕРТАЦИЯ

за присъждане на образователна и научна степен "доктор" по научна специалност „Информатика"

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

Величко Георгиев Джамбов

Ръководител: акад. Васил Сгурев

Научно жури:

Проф. Иван Димов
Акад. Васил Сгурев
Акад. Иван Попчев
Акад. Минчо Хаджийски
Акад. Петър Попиванов

Институт по информационни и комуникационни технологии

Секция „Интелигентни системи"

Дисертацията е обсъдена и допусната до защита на разширено заседание на секция „Интелигентни системи “ на ИИКТ-БАН, състояло се на ??.??. 2016 г.

Дисертацията съдържа 176 стр., в които 56 фигури, 17 таблици и 9 стр. литература, включваща 173 източника.

Защитата на дисертационния труд ще се състои на ??.??.2016 г. от ??:?? часа в зала ??? на блок ??? на ИИКТ-БАН на открито заседание на научно жури в състав:

1.
2.
3.
4.
5.

Материалите за защитата са на разположение на интересуващите се в стая 215 на ИИКТ-БАН, ул. „Акад. Г. Бончев“, бл. 25А.

Автор: ***Величко Георгиев Джамбов***

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

Увод

Настоящият дисертационен труд е посветен на изследване на възможностите за реализация на средства и методи за ефективно решаване с много висока точност на някои класове задачи в контекста на конкретна програмна среда - .Net Framework в операционна система Windows. Има се предвид създаването на инструментална програмна система с развойна цел, използваща възможностите на самата среда.

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 [51]. 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 [57].

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.

Това прави необходими 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 създаване на собствени такива инструменти в контекста на .NET Framework в операционна

система Windows. 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, по-надолу), 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.

Макар в общността на създателите на научен софтуер комерсиалните среди да не са на почит, трябва да се отчете, че конкретната среда е много широко разпространена за настолните и преносими персонални компютри. Даването на заинтересованите от решаване на 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 - 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 е създадена преди всичко 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.

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 [52] (разклонение на GMP). Всичко необходимо по генерирането на тези динамични библиотеки може да се изтегли от [47].

Ефективността изисква усиля в насоки с различно ниво на конкретност. Най-общите изисквания са за включване на методи за пресмятания с подходяща асимптотична оценка. Следва експериментално уточняване в случая на конкретни диапазони на изискваната точност. Друга възможност е използването на целочислена аритметика там, където е възможно. И разбира се, използване на възможността за паралелни пресмятания, предоставяни от конкретната среда. Тук вече съществуват множество известни схеми (вж. например [17] 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 "Използване на многоядрената архитектура" в глава 2 "Методи и средства за подобряване числеността пресмятане на някои класове определени интеграли с висока точност".

Създаването на развойна система има и поне две изисквания за удобства на използване, включващи въвеждането входните данни и визуализация на получените резултати. За първата част са предложени интерпретатор на

математически изрази и нестандартен вариант на символно диференциране, използващ обработка на самия обратен полски запис, получен от интерпретатора, а не дървовидната структура на израза. За втората част е предложена схема, използваща готова безплатна програма gnuplot, при която съответният програмен инструмент инициализира нов процес и пренасочва стандартния вход на gnuplot към поток за писане. Оттам нататък той може да извежда в gnuplot както желаните данни, така и информация, управляваща визуализацията.

Цели и задачи на дисертацията

Основната цел се състои в:

Даване на възможност на изследователи и разработчици с опит в конкретната среда да създават свой собствен софтуер, използващ пресмятания с произволна точност, специално настроен за интересуващите ги задачи.

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

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. 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. Основната причина 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 обикновени диференциални уравнения (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 задачи на числения анализ. Направени са някои улеснения, свързани с интероперативността на някои програмни инструменти, но това не е общ подход, а резултат от опит, 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.

Въпреки че разработените програмни инструменти в предлаганата система са предимно в рамките на числения анализ (впрочем условно, вж. глава 6 "Метод за определяне на целочислена зависимост и идентификация на константи. Алгоритъм PSLQ") и тук има известни ограничения. С малки изключения, не се предвиждат програмни инструменти за една и съща задача, използващи множество различни методи. Стремежът ни е към разумна "минималност". 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, 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.

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

В тази глава са изложени основните средства и методи за пресмятане с висока точност на елементарни и специални функции, използвани в предлаганата реализация на програмната система за целевата програмна среда. Това е основата, необходима за разработване на средства за решаване с висока точност 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. Основните ползвани източници са [36], [2], [12], [15], [13], [11], [25], [35], [24], както и многобройни сайтове, едни от най-полезните от които са [46], [45], [34].

1.1. Някои методи за пресмятане

В този раздел на дисертационния труд са приведени голяма част методите и алгоритмите, използвани за пресмятане на елементарни и специални функции с висока точност.

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

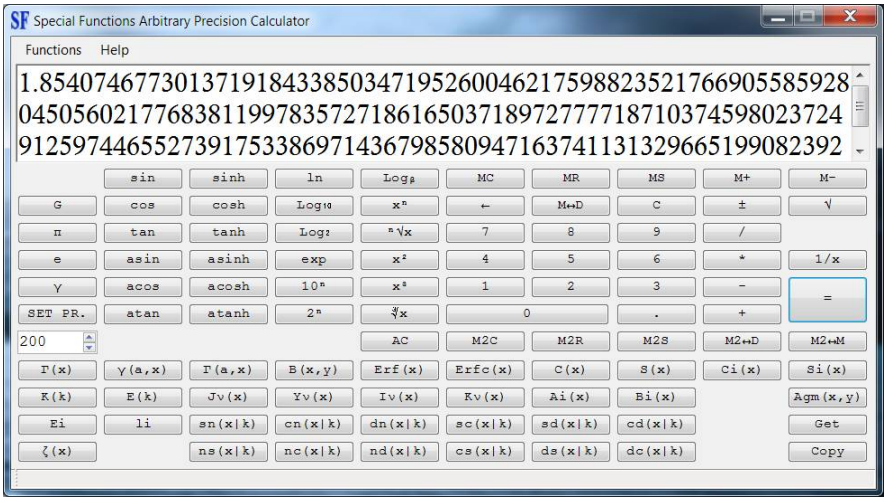
Процесът на тестване е една от най-трудоемките операции по съставяне на библиотеката с елементарни и специални функции. Основните проверки са за предварително известни стойности на дадена функция при точно определен аргумент (аргументи). Още по-добра проверка е възможността за сравняване на даден резултат, 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. 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 - програмен инструмент SFCALC. Калкулаторът позволява динамична промяна на използваната точност. 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.

Един пример за пресмятане с калкулатора при точност 200 знака:

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

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.

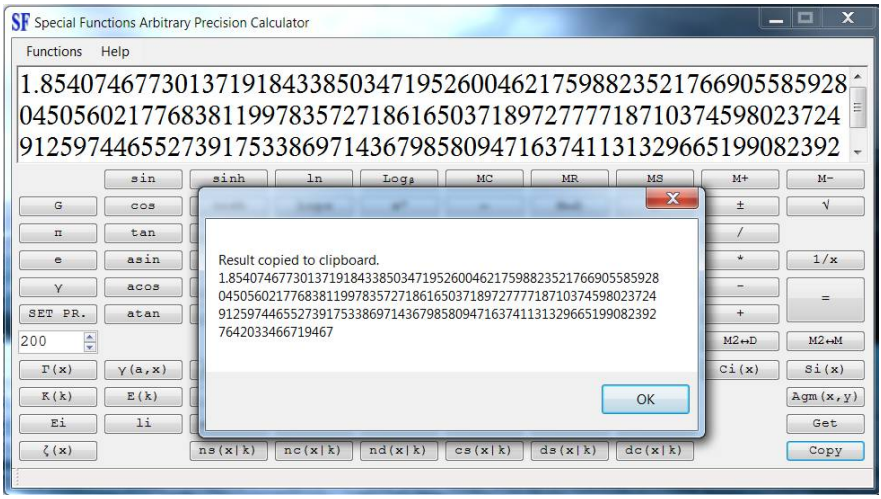
Може да се запомни с MS и след набиране на 2, $\sqrt{\ }$, 1/x, K(x) се получава лявата страна в (79) - пълен елиптичен интеграл от първи род за стойност $1/\sqrt{2}$. Не е приведена илюстрация, защото резултатът е един и същ.

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 (Фигура 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,

1.8540746773013719184338503471952600462175988235217669055859
2804505602177683811997835727186165037189727777187103745980237
2491259744655273917533869714367985809471637411313296651990823
2392764203346671946 7.

Освен това обаче в много случаи калкулаторът е изключително полезен и 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.

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.

В тази глава са 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.

Основната цел за създаване на специални програмни инструменти за пресмятането на определени интеграли с много висока точност е свързана с възможното използване на резултатите в процедура от тип идентификация на константа. Един съвременен обзор, включващ и по-съвременни методи е [48]. 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.

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.

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) [5]. В [43] са приведени точни математически резултати за класове от функции, към които са приложими различни видове квадратурни формули от тип двойна експонента. Простата евристична

обосновка на идеята посредством формулата на Ойлер-Маклорен е от сравнително скоро (вж. например [8]).

Квадратурната формула на Кленшоу-Къртис [53] 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. 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: $O\left(V(2N)^{-n}\right)$.

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.

В предлаганата програмна система е реализиран специален графичен програмен инструмент NQTS, както и два конзолни (THSHPar, CCPar). В THSHPar и CCPar се използват паралелни пресмятания. NQTS и THSHPar използват метода на "двойната експонента". CCPar използва квадратурна формула на Clenshaw-Curtis.

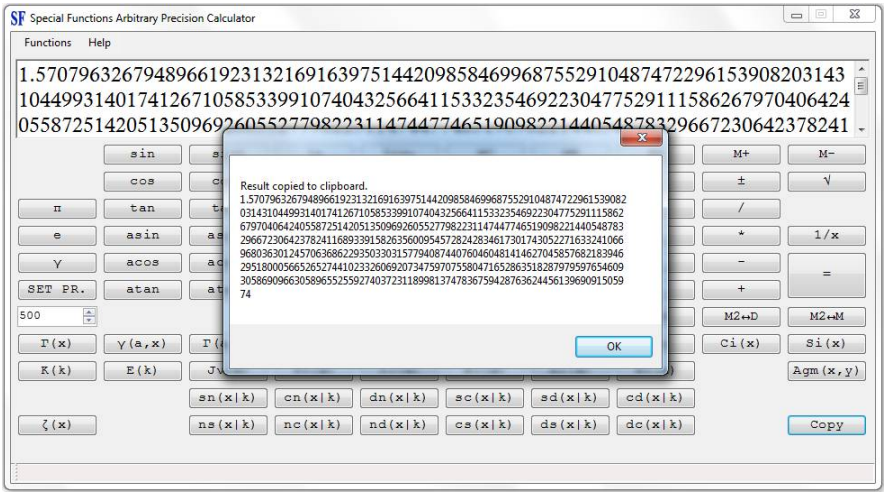
2.1. Използване на двойно експоненциално преобразование за числено пресмятане на определени интеграли

В този раздел на дисертационния труд са приведени някои сведения, необходими за извеждане на tanh-sinh формулата, както и за обяснение на нейната ефективност. Приведен е и един основен вариант на алгоритмична схема, използваща последователни нива до достигане на зададената целева точност.

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

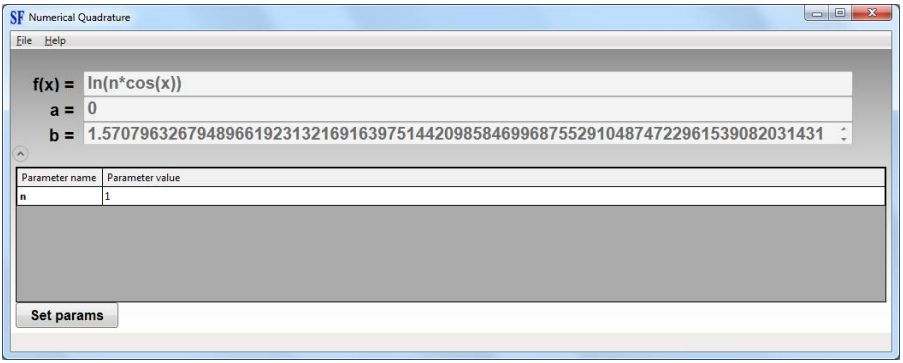
Програмният инструмент NQTS е интерактивен и графичен, предназначен за числено пресмятане на определени интеграли с висока точност. Областта на интегриране може да бъде краен или безкраен интервал. Безкрайният интервал може да е от типа (a,∞) или $(-\infty,\infty)$. Изчисленията се извършват с произволна, предварително задавана, точност. В израза за подинтегралната функция 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.

Ако за инициализирането на някакво поле или параметър е нужна определена математическа константа, може да се използва програмният инструмент SFCALC. 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 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.



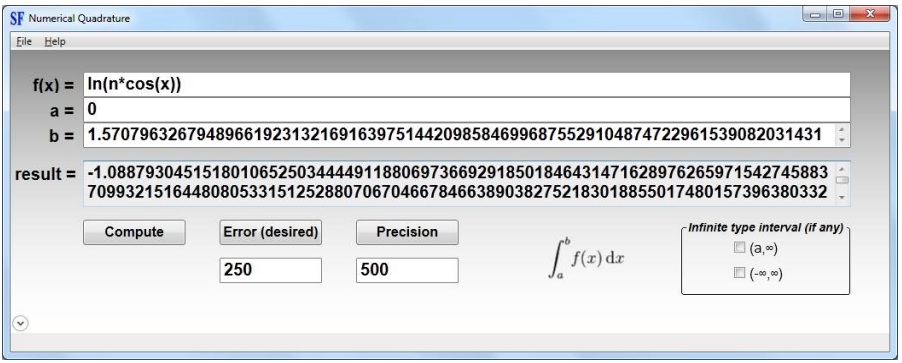
Фигура 2.6. Прехвърляне на желания резултат от SFCALC в клипборда.

Изчиства се полето за дясна граница на интервала (b) и се копира (Ctrl+V). 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

$$\int_0^{\frac{\pi}{2}} \ln(\cos(x))\,dx$$ в NQTS.

Полето на резултата съдържа

1.088793045151801065250344449118806973669291850184643147162897626597154274588370993215164480805331512528807067046678466389038275218301885501748015739638033208354306398902900085594241828542310963591445370660726023127015955425916345915993755 1

44714047**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**.

Един интересен пример [5], чиято стойност е известна, но затруднява специализирани среди за математически пресмятания с отгатването на аналитичната (затворена) форма на резултата.

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

След пресмятането на лявата част 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**.

2.3. Квадратурна формула на Clenshaw-Curtis

В този раздел на дисертационния труд са приведени сведенията, необходими за извеждане на формулата на Кленшоу-Къртис, и обяснена нейната изключителна ефективност за конкретен клас функции: гладки периодични функции, всички производни на които са с ограничена вариация.

2.4 Използване на многоядрената архитектура.

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)). За целта се предварително формира обект thsh, който 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. Използва се клас XW, чийто основен метод xw пресмята абсцисите и теглата в зададен интервал [idx1,idx2). Реализиран е доста икономично. Абсцисите и теглата са съответно $x_k = \tanh\left(\frac{1}{2}\pi\sinh(kh)\right)$ 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. 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. 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.

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. [55], 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 [56]. 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.

В дисертационния труд са приведени части от програмия код, илюстриращ горното неформално описание на реализацията, използваща многоядрената архитектура.

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 таблицата по-долу е отбелязан като TL (test laptop).

За база на изследването се взема [7]. 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 [7]. 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.

		Брой процесори			
Пример	Изисквано ниво	4, [7]	16, [7]	1024, [7]	4, THSHPar+TL
Иниц. 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 иниц.		3949.74	989,6		988.00
Корекция за хардуера в [7]	1/[(3.1/2)* 1.3]	1960.17	492,12		988.00

Таблица 2.2. Сравнителни резултати за THSHPar и набор от примери в [7].

В последния ред е добавена корекция, отчитаща максималното възможно ускорение спрямо използвания в [7] хардуер, имайки предвид максимално достигната 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 [7]. При такива данни (без включване на времето за инициализация и в двете реализации, тази в [7] 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 [7]. Без корекцията, резултатът е равен на 16 процесора от [7].

За пример 14 при 13 нива не се достигат желаните 2000 знака, а около 1971 (и в [7], и за THSHPar+TL). В [7] 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	Изисквано	4, [7]	16, [7]	64, [7]	256, [7]	1024,	4, CCPar

	ниво в [7]	ниво в CCPar					[7]	+TL
(Иниц. 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 (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 [7].

Там, където е приложима, паралелната Clenshaw-Curtis квадратура (с 4 процесора: CCPar + TL) е еквивалентна на около 128 процесора от системата използвана в [7]. 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.

В раздела за програмния инструмент NQTS бе споменат интеграла $\int_0^{\pi/2} \frac{\arcsin\left(\sqrt{2}/2 \cdot \sin x\right) \sin x}{\sqrt{4 - 2\sin^2 x}} \mathrm{d}x$, чиято аналитична оценка затруднява разпространените специализирани среди за математически пресмятания, дава резултата с точност 1000 десетични знака след пресмятането. Това

пресмятане, в онзи момент (около две и половина години преди разработването на THSHPar и CCPar) всъщност отнемаше повече от 4 часа 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.

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 вгражданите в съвременните преносими и настолни компютри многоядрени процесори.

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

В тази глава са описани възможностите на разработените програмни инструменти за пресмятане на (системи) обикновени диференциални уравнения с много висока точност.

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

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, численото 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 - [19], [16], [28], [54]). 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).

Специално обсъждане заслужават възможностите и смисълът на решаване на системи ОДУ с висока точност за случая, в който те са модел на специален вид системи пораждащи "детерминиран хаос". Примери и илюстративни решения на някои такива системи са изнесени в глава 7.

3.2. Обзор на неявните схеми на Рунге-Кута.

В този раздел на дисертационния труд е направен обзор на резултатите, касаещи неявните схеми на Рунге-Кута за решаване на системи обикновени диференциални уравнения, използвани в програмните инструменти на предлаганата система.

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.

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.
```

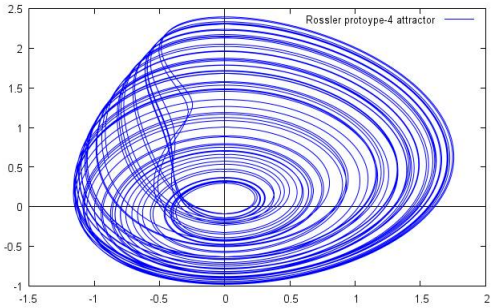
Първият аргумент от командния ред е управляващ режима (решаване, визуализация) и следващите се интерпретират според него. При решаване се задават ред на неявната схема, точност, и брой стъпки, начална точка. Резултатът е изходен файл на решението с фиксирано име irktest.dat. С това приложение, бяха осъществени редица тестове. Някои от тях са изведени в отделна глава 7. Тук ще покажем само три.

Система на Рьослер, прототип 4

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

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

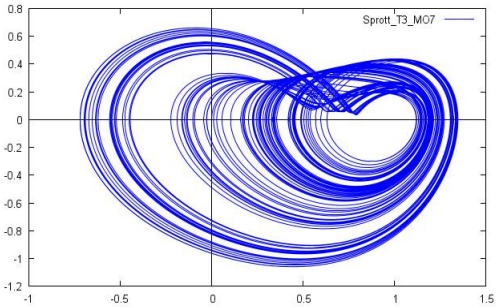
$$(x_0, y_0, z_0) = (2, 0, 1)$$



Осцилатор с памет

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

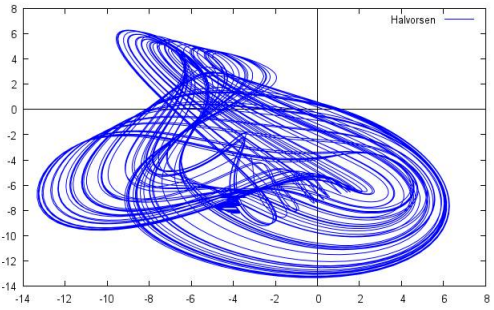
$$(\ddot{x}_0, \dot{x}_0, x_0) = (0, 0, 0.4)$$



Циклична система $(Halvorsen)$

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

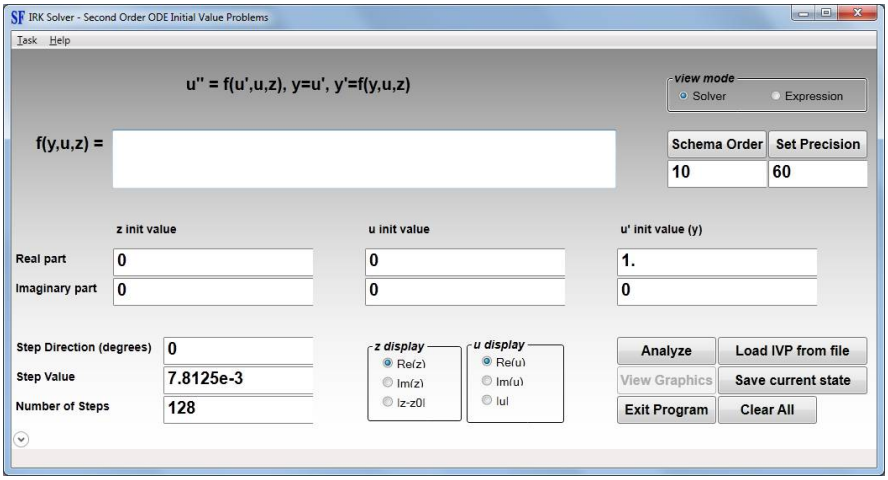
$$(x_0, y_0, z_0) = (-6.4, 0, 0)$$



Фигура 3.2. Примери за получени и визуализирани с IRK_Test решения.

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 ([29],[22]), който е адаптиран за C# + XMPIR за използване с произволна точност.

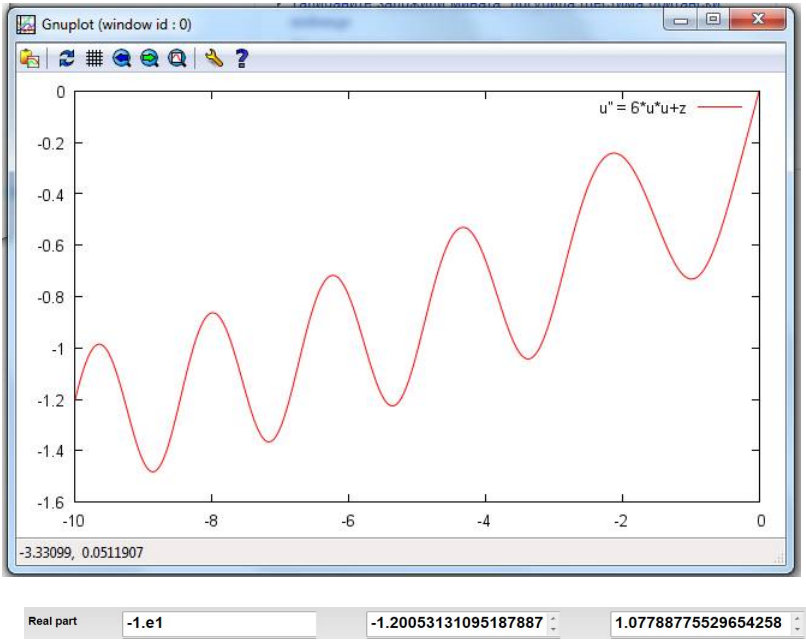
Интерактивният графичен програмен инструмент VODE2 е предназначена за числено изследване с визуализация на решението на обикновено диференциално уравнение от втори ред $\mathbf{u'' = f(u', u, z), (' = d/dz)}$. Той решава задача начални стойности $\mathbf{z = z_0}$, $\mathbf{u_0 = u(z_0)}$, $\mathbf{u'_0 = u'(z_0)}$. 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. 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. Всички пресмятания се извършват с комплексни числа (произволна точност).



Фигура 3.3. Програмата VODE2 след първоначално стартиране.

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. 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 се изисква отделна инициализация. 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. Идеята за създаване на този програмен инструмент възникна от желанието за числено изследване на решения на някои уравнения на Пенлеве.



(мнимите части за нули)

Фигура 3.5. Резултат от пресмятането на $u'' = 6u^2 + z$ с 1280 стъпки в отрицателна посока и големина на стъпката по подразбиране, но със знак минус (от 0 до -10) и всички други стойности по подразбиране.

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.

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 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, например при пресмятанията на някои константи. 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.

4.1. Методи с локална сходимост.

За случая на достатъчно добро първоначално приближение, 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 [40]. 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 [40] 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.

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). Тук ще се опишат идеите, използвани за реализирания в програмната система вариант на метод на продължението чрез хомотопия (Homotopy Continuation Method, [18], [30]). 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 [37].

Методът е в групата на така наречените методи на продължението (Numerical Continuation Methods) [3]. 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 $F(x,\lambda_0)=f(x)=0$ 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) $$H(x,t)=f(x)-(1-t)f(x_0),$$

(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 $t(s)=1$.

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). Предлаганата програмна система разполага 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 инструментални средства дават 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.

Функцията $f(x) = 2x - 4 + \sin(2\pi x)$ 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*. 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, достиганата 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.

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

4.3.1. Намиране на корените на класически ортогонални полиноми.

Основните източници на информация за този подраздел са [42], [32]. [44], [33].

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).

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.

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 с варианта на xmpir от 2010 година.

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 [23].

Схемата за пресмятане на функции на Бесел (от първи или втори род, или тяхна линейна комбинация) е изключително проста и се свежда до следното: 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 $\Delta^{(n)} = \pi/2$, в противен случай $\Delta^{(n)} = z_*^{(n-1)} - z_*^{(n-2)}$ (това дава по-добро начално приближение).

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 [36], [20].

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 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 предлаганата реализацията 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 ([36], 9.9(iv) и [20])

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

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

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

Необходими са само първите няколко точни стойности за T_n и U_n, тъй като 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.

В тази глава се обсъждат методите, лежащи в основата на разработените методи за бързо пресмятане на някои математически константи с 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 предлагания 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 [14]. По-съвременни изложения има в [27], а също и в глава 4 на [15].

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.

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

В този раздел на дисертационния труд е описана техниката на двоичното разделяне, правеща възможно използването на паралелни пресмятания на някои редове.

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 се използват 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 оригнала (за конкретния процесор, разбира се).

5.3. Сравнителни резултати за няколко константи.

За сравнение ще се използва безспорния лидер към момента - програмата γ-crunher [49]. Приведени са и някои резултати от доста старата, но най-добра за времето си програма PiFast. 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. Обзор на използваните методики, даващ представа за цялостната картина (и огромния обем код) има в [50]. 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 предлаганата система влиза използване на точност от порядък до няколко хиляди знака (за PSLQ, например това обикновено е 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.

Данните са в секунди. Лемниската (Lemniscate) в таблиците е името на константата $\left[\Gamma\left(1/4\right)\right]^2/\sqrt{2\pi}$. 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

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

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

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

5.4.2. Резултати с MPConsts.

Пресмятането на константата на Хинчин се базира на [9]. 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 [9], 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 ζ 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 [15].

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 [26] има резултат от 1998 година (Xavier Gourdon) за 110000 знака. Този резултат е записан като рекорден (в [34] 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.

5.5. Константа на Рамануджан-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 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 Рамануджан-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 [39]. 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 [21]. 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 ([31], [41])

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

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

Може да се използва факта ([21]), 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.

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

$$\begin{aligned}\ln(\lambda\sqrt{2}) &= \sum_{n=1}^{\infty}\frac{1}{2^{n+1}}\ln\prod_r\frac{1+r^{-2^n}}{1-r^{-2^n}}=\sum_{n=1}^{\infty}\frac{1}{2^{n+1}}\sum_r\ln\left(\frac{1+r^{-2^n}}{1-r^{-2^n}}\right)=\\ &=\sum_{n=1}^{\infty}\sum_r\frac{1}{2^{n+1}}\ln\left(\frac{1+r^{-2^n}}{1-r^{-2^n}}\right)=\sum_r\sum_{n=1}^{\infty}\frac{1}{2^{n+1}}\ln\left(\frac{1+r^{-2^n}}{1-r^{-2^n}}\right)\end{aligned} \tag{5.17}$$

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 [21] 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

$$\frac{xe^{\sqrt{1+x^2}}}{1+\sqrt{1+x^2}} - 1 = 0 .$$

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.

Тази глава е посветена на разработката на метод за ефективна програмна реализация на алгоритъма за определяне на целочислена зависимост PSLQ (Partial Sums using LQ decomposition) в неговия основен вариант за настоящата програмна среда.

Определянето (откриването) на целочислена зависимост (integer relation detection) решава следната задача: при дадени *n* реални числа x_1, x_2, ... , x_n да се открие съществуват ли *n* цели числа (не всички равни на 0) a_1, a_2, ... , a_n, за които $a_1x_1 + a_2x_2 + ... + a_nx_n = 0$. Това равенство, разбира се, при пресмятания с компютър е с точност до "епсилона на използваната точност".

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

В този раздел на дисертационния труд е описан базовият вариант на алгпритъма 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).

6.2. Практическа реализация.

PSLQ съществува в два основни варианта - базов и така наречения "multi-pair" ([10], 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).

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 инструменти за пресмятане с много висока точност на определени интеграли, както и реализацията за бързо пресмятане на константи, чийто списък очевидно би трябвало да се разширява.

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.

Интересен и по-реалистичен 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, $4913 + 2108t^2 - 604t^3 - 977t^4 + 8t^5 + 44t^6 + 392t^7 - 193t^8 - 40t^9 + 48t^{10} - 12t^{11} + t^{12}$, чийто корен е

$r = B_3 =$
3.544090359551922853615965986604804540583099845444573675457812530305842942858863012256258566424891799962 6... (третата бифуркация на логистичното изображение, [6]).

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).

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.

В тази глава на дисертационния труд са приведени са примерни резултати от практическо решаване с висока точност на системи с "хаотично поведение", получени с разработените в настоящата работа програмни инструменти. Първият раздел съдържа илюстрация към обсъждането в раздел 3.1 от глава 3 за възможностите за количествено изследване при решаване с все по-голяма точност. Вторият раздел съдържа примери с илюстрации на решения на прости гладки динамични системи с хаотично поведение.

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. Един от тях е онагледяване на възможностите за количествено изследване при решаване с все по-голяма точност. Останалите са илюстрации 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.

В първия раздел на тази глава от дисертационния труд е представен типичен програмен код, реализиращ паралелни пресмятания с помощта на двоично разделяне. Във втория раздел са приведени представяния на някои често използвани математически константи във вид на редове, , които са подходящи за прилагане на двоично разделяне.

8.3. Заключение.

Приведена е илюстрация на типичен за представяната система 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, даващи възможност за прилагане на двоично разделяне.

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. Описани са редица числени примери, които показват ефективността на предложените в дисертационния труд методи и алгоритми, както и възможностите за тяхната практическа реализация.

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] Allgower E., Georg, K., "Numerical Continuation Methods", Springer-Verlag, 1990.

[4] Bailey, D. H., "Experimental Mathematics and Optimization", invited short course presentation, McMaster University, Canada, Aug 2007.

[5] Bailey, D. H., "Peter Borwein and High-Performance Computing", 2008, http://crd-legacy.lbl.gov/~dhbailey/dhbtalks/dhb-peter-borwein.pdf.

[6] 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.

[7] Bailey, D., J. Borwen, "Highly Paralell, Highly-Precision Numerical Integration", 2008, http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/quadparallel.pdf.

[8] 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.

[9] Bayley, D., J. Borwein, R. Crandall, "On the Khintchine Constant", Mathematics of Computation, vol. 66, 1997, 417–431.

[10] 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.

[11] Bateman, H., A. Erdélyi, "Higher Transcendental Functions" Vol. 1, 2, 3, McGraw-Hill Book Company, 1953-1955.

[12] 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.

[13] 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.

[14] 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.

[15] Brent, R. P, "Modern Computer Arithmetic", Cambridge University Press, 2011.

[16] Butcher, J. C. "Numerical Methods for Ordinary Differential Equations", Second Ed. John Wiley & Sons, Ltd., 2008.

[17] 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.

[18] 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.

[19] Dekker, K., J. G. Verwer. "Stability of Runge–Kutta Methods for Stiff Nonlinear Differenrial Equations", North-Holland, 1984.

[20] 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.

[21] Flajolet, P., Vardi, I., "Zata Function Expansions of Classical Constants", 1996, http://algo.inria.fr/flajolet/Publications/FlVa96.pdf.

[22] Fortran and Matlab Codes, http://www.unige.ch/~hairer/software.html.

[23] 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.

[24] 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.

[25] Gil, A., J. Segura, N. M. Temme. "Numerical Methods for Special Functions" SIAM, 2007.

[26] Gourdon, X., https://github.com/jlapeyre/mext/blob/master/packages/discrete_aex/khintchine.txt

[27] 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.

[28] Hairer, E., C. Lubich, G. Wanner, "Geometric Numerical Integration, Structure- Preserving Algorithms for Ordinary Differential Equations", Second Edition, Springer, 2006.

[29] Hairer, E., S.P. Nørsett, G. Wanner, "Solving Ordinary Differential Equations I, Nonstiff Problems", Second Revised Edition, Springer, 1993.

[30] Judd K., "Numerical Methods in Economics", The MIT Press, Cambridge, Massachusetts, London England, 1998.

[31] 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.

[32] McNamee J., "Numerical Methods for Roots of Polynomials - Part I", Elsevier, 2007.

[33] McNamee J.M., Pan V.Y., "Numerical Methods for Roots of Polynomials - Part II", Elsevier, 2013.

[34] Numbers, constants and computation (site, Xavier Gourdon and Pascal Sebah), http://numbers.computation.free.fr/Constants/constants.html.

[35] Olver, F. W. J., "Asymptotics and Special Functions", Academic Press, 1974.

[36] 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.

[37] 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.

[38] Segura, J., "The zeros of special function from a fixed point method", SIAM J. Numer. Anal., 40 (2002), 114-133.

[39] Shanks, D., "The second-order term in the asymptotic expansion of B(x)", Math. Comp. 18 (1964) 75–86.

[40] 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.

[41] Srivastava, H. M., Choi, J., "Zeta and q-Zeta Functions and Associated Series and Integrals", Elsevier, 2012.

[42] Szegö G., "Orthogonal Polynomials", American Mathematical Society, 1939.

[43] Tanaka, K., M. Sugihara, K. Murota, M. Mori, “Function Classes for Double Exponential Integration Formulas”, Methematical Engineering Technical Reports, Tokyo University, 2007.

[44] Volpi L., "Non Linear Equations: Practical Methods for Root Finding, Part II - Zeros of Polynomials", Foxes Team, 2006.

[45] Wikipedia site, http://en.wikipedia.org/
[46] Wolfram Functions site, http://functions.wolfram.com/.

[47] XMPIR, http://www.alglib.net/x/xmpir/xmpir-0.2.zip.

[48] 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

[49] Yee, A., http://www.numberworld.org/y-cruncher/#Download, 2015.

[50] 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.

[51] 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

[52] http://mpir.org/

[53] Clenshaw C. W. and Curtis A. R., "A method for numerical integration on an automatic computer", Numerische Mathematik 2, 197-205 (1960)

[54] Kang Feng, Mengzhao Qin, Symplectic Geometric Algorithms for Hamiltonian Systems, Springer, 2010.

[55] Arndt J., "Matters Computational", Springer, 2011.

[56] Waldvogel J., "Fast Construction of the Fejér and Clenshaw–Curtis Quadrature Rules", BIT Numerical Mathematics, March 2006, Volume 46, Issue 1, pp 195-202.

[57] Robey R. W., Robey J. M. and Aulwes R., "In search of numerical consistency in parallel programming," Parallel Computing, vol. 37 (2011), 217-219.

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.

Abstracts of Dissertations

Number 1, 2017

INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGIES

BULGARIAN ACADEMY OF SCIENCES

БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ

ИНСТИТУТ ПО ИНФОРМАЦИОННИ И КОМУНИКАЦИОННИ ТЕХНОЛОГИИ

Брой 1, 2017

Автореферати на дисертации