BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **14**, No 1

Sofia • 2014

Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2014-0014

High Precision Computing of Definite Integrals with .NET Framework C# and X-MPIR

Velichko Dzhambov

Institute of Information and Communication Technologies, 1113 Sofia
E-mail: vili_jambov@abv.bg

Abstract: *This paper concerns high precision numerical computing of definite integrals in a specific environment, namely .NET Framework. The work is a part of a series, tracing the progress of creating tools for high precision computations in this environment and may be considered as a continuation, in this direction, of the beginning, described in [15], that includes special function calculations with arbitrary precision. Some of the methods used are described. The results are clearly illustrated with the help of an application, purposefully created, using the current state-of-the-art library being created for realization of functions and numerical methods of arbitrary precision in a given environment.*

Keywords: *High precision computation, numerical quadrature, definite integral computation, computational mathematics, .NET Framework.*

1. Introduction

Arbitrary precision computations are not a self-purpose. They are related to receiving precise values when solving mathematical models in different areas, including, for instance, integer relation detection to identify some definite integrals as an analytic expression of mathematical constants (“closed form”). In [15] a high precision computational library is considered in the environment of .Net Framework and some numerical analysis tools are described, as well as an auxiliary and illustrative application, based on this library.

One of the goals is to demonstrate that useful and quite powerful mutually supporting computational instruments for solving non-trivial problems may be realized in a certain environment, which is sufficiently wide spread on personal desktops or portable computers, but not enough rated by authors creating software for scientific applications.

The choice of the method for the particular problem is very important in the case considered (numerical integration of definite integrals), that is the so called double exponential transformation. It is not supposed to be the only appropriate method in all the cases. A given scheme may be applied for a given class of problems (specific features of the functions being integrated), maximally utilizing the specifics of the class. At present the library has all tools for realization of Gaussian quadrature (tested for classical orthogonal polynomial roots to degree up to 25 000 and precision up to 10^{-200}) and they are really used in other areas, e.g., systems of ordinary differential equations according to the implicit Runge-Kutta scheme. This selection is partially implied by the fact that "Tanh-sinh quadrature is the best integration scheme for functions with vertical derivatives or blow-up singularities at endpoints, or for any function at very high precision (> 1000 digits)" [16], although Gaussian quadrature also deserves attention in the cases when the function being integrated is of "good behavior" (with the necessary number of derivatives, for instance) in the whole closed interval. The cost of computing the abscissas and the weights of Tanh-sinh quadrature scheme when very high precision is required, grows linearly with the required precision, while in the Gaussian scheme it grows in a quadratic way. This is namely what makes the Tanh-sinh quadrature a more appropriate candidate under the requirement for precision of hundreds and thousands of decimal digits. Comparative analysis of three numerical quadrature schemes is given in [7].

2. Using double-exponential transformation for numerical computing of definite integrals

2.1. Euler-Maclaurin formula and the Tanh-sinh transformation

In order to introduce the method used, some preliminary explanations are needed. Let a finite interval (a,b) be given, as well as the positive integer numbers *m* and *n*. We set $h=(b-a)/n$ and $h_j = a + jh$ (and B_{2i} for Bernoulli numbers). From Euler-Maclaurin formula, assuming that the function has at least 2*m*+2 continuous derivatives, it follows

$$\int_a^b f(x)\,dx = h\sum_{j=0}^{n} f(x_j) - \frac{h}{2}(f(a)+f(b)) - \\ -\sum_{i=1}^{m} \frac{h^{2i}B_{2i}}{(2i)!}\left(f^{(2i-1)}(b) - f^{(2i-1)}(a)\right) - E(h,m), \tag{1}$$

where

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

for some $\xi\in(a,b)$, and one can see that if the function f has derivatives of any order and all of its derivatives are zero at *a* and *b* (as in a bell curve), the error member $E(h,m)$ tends to zero faster than any power of *h* (Fig. 1).

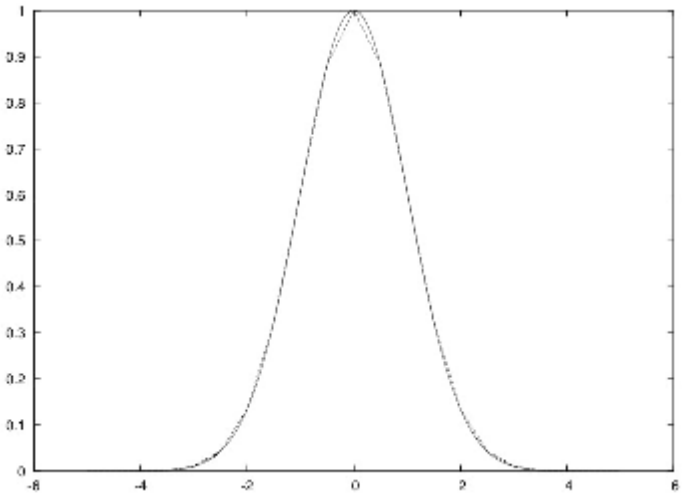

Fig. 1. A trapeze rule for the bell curve

The Euler-Maclaurin formula may be considered as giving correction of higher order for the trapeze rule.

Let for a given function f, defined in $(-1,1)$, we set $g(t)=\tanh\left(\frac{\pi}{2}\sinh(t)\right)$ with $x=g(t)$. Then

(3) $$I=\int_{-1}^{1}f(x)\,dx=\int_{-\infty}^{\infty}f\left(g(t)\right)g'(t)\,dt\approx h\sum_{j=-N}^{N}w_j f(x_j)=I_h^N,$$

where $x_j=g(hj), w_j=g'(hj)$. Since $g'(t)$ tends to zero very fast with the increase of $|t|$, the product $f\left(g(t)\right)g'(t)$ is generally a function with a well expressed bell shape. For such functions, as mentioned in the previous paragraph, the Euler-Maclaurin formula ensures that the sum above is an extraordinarilly exact approximation. Usually dividing *h* by two double, the number of exact digits (if the function is analytic in the open interval and we set $I_h=I_h^\infty=h\sum_{j=-\infty}^{\infty}w_j f(x_j)$, then

$$|I-I_h|\approx\exp\left(-\frac{c_1}{h}\right)\text{ and }\left|I-I_h^N\right|\approx\exp\left(-c_2\frac{N}{\ln N}\right).$$

The Tanh-sinh quadrature formula is called also double exponential for an evident reason – the way the derivative $g'(t)$ of the transformation decreases in infinity. This is the best integration scheme for functions with singularities at the ends of the interval of type explosive increasing and vertical asymptotes. Of course, for guaranteed good behavior of the scheme, there are requirements that the function must satisfy. For strict mathematical results for the function classes, different kinds of double exponential quadrature formulas are applied [3]. The original source is [1]. A very extensive overview of the story of the double exponential formula discovering and the further development and applications of the ideas in various areas may be found in [2]. The simple heuristic ground of the idea via Euler-Maclaurin formula is dated quite recently [8, 9]. Surprisingly, a constellation of eminent mathematicians of the past, who devoted time and efforts in the numerical quadrature area, has not come to this idea. Furthermore, many years have passed after this method publication, before it became acknowledged and used in various packages and environments for numerical analysis, which include numerical quadratures [2].

2.2. Double exponential formula variations

The formula with the Tanh-sinh transformation cited in the previous section is for the case of the finite interval $(-1,1)$. The generalization for any finite interval (a,b) is obvious, through the linear transformation

$$\int_a^b f(t)\,dt = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{a+b}{2}+\frac{b-a}{2}x\right)dx\,. \tag{4}$$

There are similar transformations for other cases, described in the table below.

Table 1. Different cases of double-exponential transformation

Type of integral	Transformation
$\int_{-1}^{1} f(x)\,dx$	$x=\tanh\left(\frac{\pi}{2}\sinh(t)\right)$
$\int_{0}^{\infty} f(x)\,dx$	$x=\exp\left(\frac{\pi}{2}\sinh(t)\right)$
$\int_{-\infty}^{\infty} f(x)\,dx$	$x=\sinh\left(\frac{\pi}{2}\sinh(t)\right)$
$\int_{0}^{\infty} f_1(x)e^{-x}\,dx$	$x=\exp\left(t-\exp(-t)\right)$

The resulting schemes have similar properties. However, the abscissas and the weights are different and must be calculated separately. Often this additional calculation may be avoided by using, for example

(5) $$\int_0^{\infty} f(t)\,\mathrm{d}t = \int_0^{1} \left(f(x) + f(1/x)/x^2 \right) \mathrm{d}x\,.$$

2.3. Algorithm for the Tanh-Sinh formula

As abscissas and weights do not depend on the function and are uniformly distributed with a step h, an efficient implementation is possible by using several levels of dividing the step into halves. They are computed for the minimal provided step ($h = 2^{-m}$ for some "level" m), then they are visited by levels and a sum is calculated for each level by computing the function for the points with a step 2^{m-k}. The sum calculated for a given level is then used for the next level. If the last two sums differ less than the desired accuracy, the process terminates. Of course, with such organization, an appropriate number of levels must be provided, that is such a minimal step (level m), which allows the desired accuracy. Twelve levels are sufficient for most integrals for precision of 1000 digits in most of the cases [4].

Below a basic algorithmic scheme in a pseudo code is given, see [9] for more details, the algorithm is described also in [6].

Input: number of levels m, function f.
Output: the approximation hS
Initialization:
$h := 2^{-m}$
for k := 0 to $20 \cdot 2^m$ **do**
  $t := kh$
  $x_k := \tanh(\pi/2 \cdot \sinh(t))$
  $w_k := \pi/2 \cdot \cosh(t)/\cosh^2(\pi/2 \cdot \sinh(t))$
  if $|x_k - 1| < \varepsilon$ **then**
    exit **do**
  end if
end for
$n_t := k$ (the value of k at exit)

Quadrature:
$S := 0$, $h := 1$
for k := 1 to m (or until the desired accuracy is obtained) **do**
  $h := h / 2$
  for i := 0 to n_t step 2^{m-k} **do**
    if mod(i, 2^{m-k+1}) $\neq$ 0 or k = 1 **then**
      if i = 0 **then**
        $S := S + w_0 f(0)$
      else
        $S := S + w_i \left(f(-x_i) + f(x_i) \right)$
      end if

end if
end for
end for
Result = *hS*

This basic scheme enables some variations, in which, for example the termination condition for the initialization is not $|x_k - 1| < \varepsilon$, but $w_k < \varepsilon^2$ and at the same time the computations are performed with precision ε^2. This particular modification is used in NQTS (the illustrative application, described below – Numeric Quadrature with Tanh-Sinh transformation), although being slower, it demonstrates exclusive stability, guaranteeing the total required precision if the necessary number of "levels" is provided.

2.4. Special transformations in the case of oscillatory functions (Fourier-type integrals)

Transformations of the type described above do not work well for integrals of the type

$$I_s = \int_0^\infty f_1(x)\sin(\omega x)\,dx\,, \tag{6}$$

$$I_c = \int_0^\infty f_1(x)\cos(\omega x)\,dx \tag{7}$$

and for other similar transformations with slow decreasing oscillatory functions, for example Bessel functions; see [11] for current presentation of the problem for high precision computing of definite integrals with oscillatory functions.

In [12] a transformation of the variable is proposed, which is appropriate for integrals of this type. A function $g(t)$ is chosen with the following properties

$$\lim_{t\to-\infty} g(t) = 0, \quad \lim_{t\to\infty} g(t) = \infty\,, \tag{8}$$

and

$$\begin{cases} \lim\limits_{t\to-\infty} g'(t) = 0, \\ \lim\limits_{t\to\infty} g(t) = t, \end{cases} \tag{9}$$

where in the last two conditions the process converges as a double exponent. The transformation is respectively

$$x = Mg(t)/\omega \text{ for } I_s, \tag{10}$$

$$x = Mg\left(t - \frac{\pi}{2M}\right)\Big/\omega \text{ for } I_c, \tag{11}$$

M is a constant, defined below in an appropriate way: by increasing x in the positive direction, the points of the formula tend to the zeroes of $\sin(\omega x)$ or $\cos(\omega x)$,

respectively as a double exponent, so that it is not necessary to compute the function values for large *x*.

The transformation proposed in [12] satisfies the conditions above mentioned and is of the form

$$g(t) = \frac{1}{1-\exp(-k\sinh t)}. \tag{12}$$

A more efficient transformation is proposed in [13]:

$$\begin{cases} g(t) = \dfrac{t}{1-\exp(-2t-\alpha(1-e^{-t})-\beta(e^{t}-1))}, \\ \beta = \dfrac{1}{4}, \quad \alpha = \beta\sqrt{1+M\ln(1+M)/4\pi}. \end{cases} \tag{13}$$

Applying the transformation $g(t)$ for I_s, we get

$$I_s = \int_{-\infty}^{\infty} f_1(Mg(t)/\omega)\sin(Mg(t))g'(t)/\omega\, dt. \tag{14}$$

Then we apply the trapeze rule with a uniform step *h* and obtain

$$I_{s,h}^{N} = Mh\sum_{k=N_-}^{N_+} f_1(Mg(kh)/\omega)\sin(Mg(kh))g'(kh)/\omega. \tag{15}$$

Similar operations are done for I_c . We choose *M* and *h* satisfying the condition

$$Mh = \pi \tag{16}$$

and we get for I_s and I_c ,respectively for large *k*

$$\begin{cases} \sin(Mg(kh)) \approx \sin(Mkh) = \sin(\pi k) = 0, \\ \cos\left(Mg\left(kh - \dfrac{\pi}{2M}\right)\right) \approx \cos\left(Mkh - \dfrac{\pi}{2}\right) = \cos\left(\pi k - \dfrac{\pi}{2}\right) = 0, \end{cases} \tag{17}$$

so that with the increase of *k*, the points tend to the zeroes of $\sin(\omega x)$ or $\cos(\omega x)$ as a double exponent.

3. Why is it necessary to compute definite integrals with high precision?

In the last decades the high precision computation of definite integrals promises to become very useful in the experimental mathematics area. In many cases it is possible to recognize in an analytic form the value of a given definite integral under the condition that this value can be computed with high precision. Usually to do this, an integer relation detection algorithm is used. Integer relation detection algorithm is a numerical algorithm, in which, given the condition $a_1x_1 + a_2x_2 + \ldots + a_nx_n = 0$ with certain real numbers x_i can recover the integer coefficients a_k, (not all zeros) or else determine that there are not any integers less than a certain size. The most frequently used algorithm for integer relation detection

is “PSLQ” algorithm (named one of the ten “algorithms of the century” by *Computing in Science and Engineering*). The name comes from using “Partial Sums” and ”LQ factorization”; see [10] for description of the algorithm and its various modifications and applications. The algorithms require the computations to be carried out with at least $d \times n$ digits, where d is the size in digits of the largest of integers a_k. Here a simple illustrative example [6] is given. The integral

$$\int_0^1 \frac{t^2 \ln t}{(t^2-1)(t^4+1)}\,dt \tag{18}$$

is considered. Its value with precision of 101 digits is

0.18067126259065494279230812898167161533711457101829676626624079429375856622413300177089825415048379 97.

Then it is possible to apply a scheme for integer relation detection (with PSLQ), which gives the result in a “closed-form”

$$\int_0^1 \frac{t^2 \ln t}{(t^2-1)(t^4+1)}\,dt = \frac{\pi^2(2-\sqrt{2})}{32}. \tag{19}$$

Below it is shown how the result looks like in NQTS in several seconds.

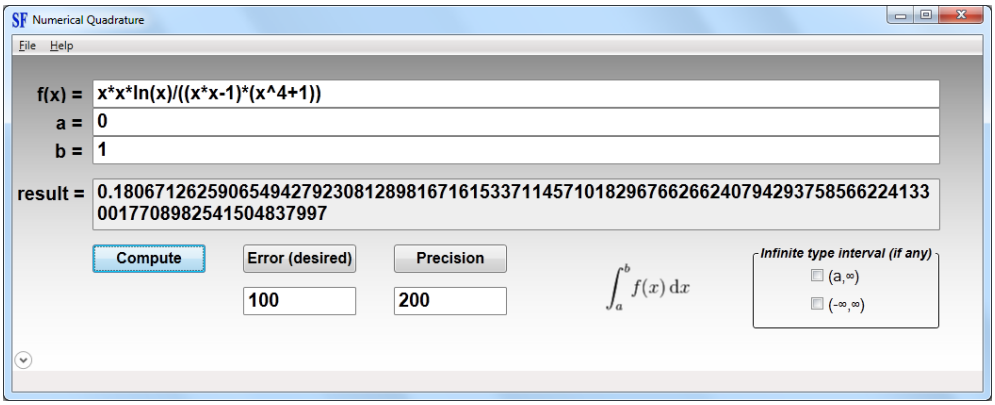

Fig. 2. View of NQTS application

As above mentioned, this is a necessary step for creating an implementation of a scheme for recognition based on PSLQ in the specific environment.

4. Purpose and capabilities of NQTS application

NQTS application is designed for numerical computing of definite integrals with high precision. The domain of the integration can be a finite or infinite interval. The infinite interval can be of the type (a, ∞) with an arbitrary real number a or $(-\infty, \infty)$. The calculations are done with an arbitrary (given by the user) precision. The expression of the function to be integrated is given in a text form and may include algebraic operations and functions (elementary and special). Additional parameters are allowed in the expression.

4.1. Resources and methods used

The arbitrary precision of the calculations (with real numbers) is provided by X-MPIR library. X-MPIR ensures the link from C# to MPIR (for details search the Web, there is no separate site, devoted to X-MPIR at this moment). On the basis of X-MPIR, which provides the basic arithmetic operations only, a library of elementary and special functions with arbitrary precision is implemented. The problem is solved with the so called double-exponential transformation – see the previous Section 2. The user chooses the desired result precision. For computation of the function to integrate, an interpreter is provided. The interpreter recognizes the mathematical expressions which include the functions in the specially created library. The interpreter accepts any number of variables, but the ones that are different from the main variable of the problem (x) are considered as parameters, for whose initialization a separate setting is required from the user.

4.2. Functional properties, special features and limitations

The program is implemented using C# in .Net Framework. It is built, targeting the maximal possible portability (32- or 64-bit Windows systems). No installation is needed. The necessary files are copied in a separate folder. Any intermediate calculations are done with the given current precision, which is the double of the desired precision for the result. For all apriori needed data concerning the problem solving, the corresponding elements of the interface are provided. The formal correctness is verified. The variable name for the variable of the function to be integrated is fixed (x) to avoid useless complications while formulating the problem. Besides this variable however, other variables can be present in the expression (parameters). When parameters are present, the program requires from the user to initialize them separately. The basic limitation is the type of the problem being solved. It is for a real valuated function, defined within an interval of the real line. *This version of the program does not include calculation of definite integrals with **slowly** decreasing oscillatory functions in infinite intervals*. At the moment it is possible to include solving of a particular kind of such problems $f(x) = f_1(x)\sin(\omega x)$ or $f(x) = f_1(x)\cos(\omega x)$, where $f_1(x)$ is not an oscillatory function with a transformation, proposed by Ooura (see Section 2). Furthermore, *no optimizations are implemented for parallel computations, using the eventual multiple core processor architecture*. This is a task that requires a change in the scheme of the algorithm and is expected to be done. One scheme is described in [5], but different tools for parallel computing must be used in the present environment. Putting aside the fact that the calculation of certain types of definite integrals is an alternative way to compute specific functions, probably the most important reason to compute definite integrals with high precision consists in the possibility to use the result from the algorithm, recognizing mathematical constants, for example PSLQ [10], and the related with this discovery still unknown relations.

The program was tested on various examples of moderate precision, up to several hundred digits, including the 14 examples, listed in [5].

At the end an interesting example from [16] will be considered, whose exact analytical value is known, but it is not easy to receive it automatically even in some special environments for mathematical computations.

The integral

$$\int_0^{\pi/2} \frac{\arcsin\left(\sqrt{2}/2 \cdot \sin x\right) \sin x}{\sqrt{4 - 2\sin^2 x}}\, dx \;=\; \frac{\sqrt{2}\pi \ln 2}{8} \tag{20}$$

calculated with the help of NQTS for the required precision of 1000 decimal digits, the calculations being carried out with 2000 digits precision gives

0.384946472767794677379733634534350939378637085633991860421625207172
215528941947536325804048608866852784932153323831795770777237037064 0
045192932824251457528736348665950631823509922239349846177129846349 3
768140745660686666596850906710592791288018356740777856332046528241 0
995242877178661051292043958165351915619510255649777686054433077543 1
426535928105430436171994118917554382798182139361249998028849475821 6
706401278779860100503585410049816722109008711289444547198386034436 8
649991070516232661145273551270231546458390444039394554059488480527 1
540730621291118431400604734284519117422288943648355891480620110477 4
482649701231546039017621206500465025993783687415897293765701988613 6
020915118403389004952631067906267002699417228747182612156794411131 5
202455208217249381660735387428913405457298197463226830060990584807 3
796563894057296570056861272215974026045319879192931935339688204716 6
537212420238993969015728751752761019410198235455831083984882671587 9
960306667055889614053090753005875188213529520407071003093521683.

NQTS shows one digit more than the required precision. The value calculated from the right hand side of the expression in (20) with SFCALC, precision of 1001 digits, differs in the last decimal digit – 2 instead of 3. The calculation took more than 4 and a half hours, including the primary initialization of about 55 minutes on the author's laptop.

5. Conclusion

A limited but representative part, concerning the numerical quadrature of high precision is presented as part of the realization of computational tools and a library for calculations of arbitrary precision in a non-typical environment, namely .Net Framework. The combination of well-considered methods plus the excellent possibilities for visualization and the human interaction in this environment is worth the efforts and provides the possibility to achieve non-trivial results in home conditions. Possibilities exist for further improvements in various directions, including the usage of the multi-core architecture of modern processors for parallel processing. In this way a wider audience would be interested, since traditionally similar software products are developed under Unix-like systems equipped with specialized hardware.

Acknowledgments: The research work reported in the paper is partly supported by the project AComIn "Advanced Computing for Innovation", Grant 316087, funded by FP7 Capacity Programme (Research Potential of Convergence Regions).

References

1. Takahasi, H., M. Mori. Double Exponential Formulas for Numerical Intergration. – Publications of RIMS, Kyoto University, Vol. **9**, 1974, 721-741.
2. Mori, M. Discovery of Double Exponential Transformation and its Developments. – Publications of RIMS, Kyoto University, Vol. **41**, 2004, 897-935.
3. Tanaka, K., M. Sugihara, K. Murota, M. Mori. Function Classes for Double Exponential Integration Formulas. Methematical Engineerong Technical Reports, Tokyo University, 2007.
4. Bailey, D. Tanh-Sinh High-Precision Quadrature. 2006. **http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/dhb-tanh-sinh.pdf**
5. Bailey, D., J. Borwen. Highly Paralell, Highly-Precision Numerical Integration. 2008. **http://crd.lbl.gov/~dhbailey/dhbpapers/quadparallel.pdf**
6. 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.
7. Bailey, D., K. Jeyabalan, X. Li. A Comparison of Three High-Precision Quadrature Schemas. – Experimental Mathematics, Vol. **3**, 2005, 317-329.
8. Bailey, D. H., J. M. Borwein, N. J. Calkin, R. Girgensohn, D. R. Luke, V. H. Moll. Experimental Mathematics in Action. Wellesley, MA, A. K. Peters, Ltd., 2007.
9. Borwein, J., D. Bailey, R. Girgensohn. Experimentation in Mathematics, Computational Paths to Discivery. A. K. Peters, Ltd., 2004.
10. Bailey, D., D. Broadhurst. Parallell Integer Relation Detection: Techniques and Applications. – Mathematics in Computation, Vol. **70**, 2000, No 236, 1719-1736.
11. Bailey, D., J. Borwein. Hand-on-Hand Combat with Thousand-Digit Integrals. – Journal of Computational Science, Vol. **3**, May 2012, Issue 3, 77-86.
12. Ooura, T., M. Mori. The Double Exponential Formula for Oscillatory Functions over Half Infinite Interval. – J. Comp. Apl. Math., Vol. **38**, 1991, 353-360.
13. Ooura, T. A Double Exponential Formula for Fourier Transforms. – Publications of RIMS, Kyoto University, Vol. **41**, 2005, 971-977.
14. Bailey, D. H. Experimental Mathematics and Optimization. Invited Short Course Presentation, McMaster University, Canada, August 2007.
15. Dzhambov, V., S. Drangajov. Computing of Special Functions with Arbitrary Precision in the Environment of .NET Framework. – Cybernetics and Information Technologies, Vol. **11**, 2011, No 2, 32-45.
16. Bailey, D. H., P. Borwein. High-Performance Computing. **http://crd-legacy.lbl.gov/~dhbailey/dhbtalks/dhb-peter-borwein.pdf**