CALCULATION OF SPECIAL FUNCTIONS WITH ARBITRARY PRECISION IN THE .NET FRAMEWORK ENVIRONMENT

Aynakulov Toxir Turg'un o'g'li
aynaqulovtohir@gmail.com
Norqo'ziyev Quvonchbek Komiljon o'g'li
quvonchbek9535@gmail.com
Irgasheva Umida Abdimital qizi
irgashevaumida1997@gmail.com
Teachers of the Jizzakh branch of the National University of Uzbekistan
Abdurashidova Lobar Ulug'bek qizi
Abdurashidova0809@gmail.com
Student of the Jizzakh branch of the National University of Uzbekistan

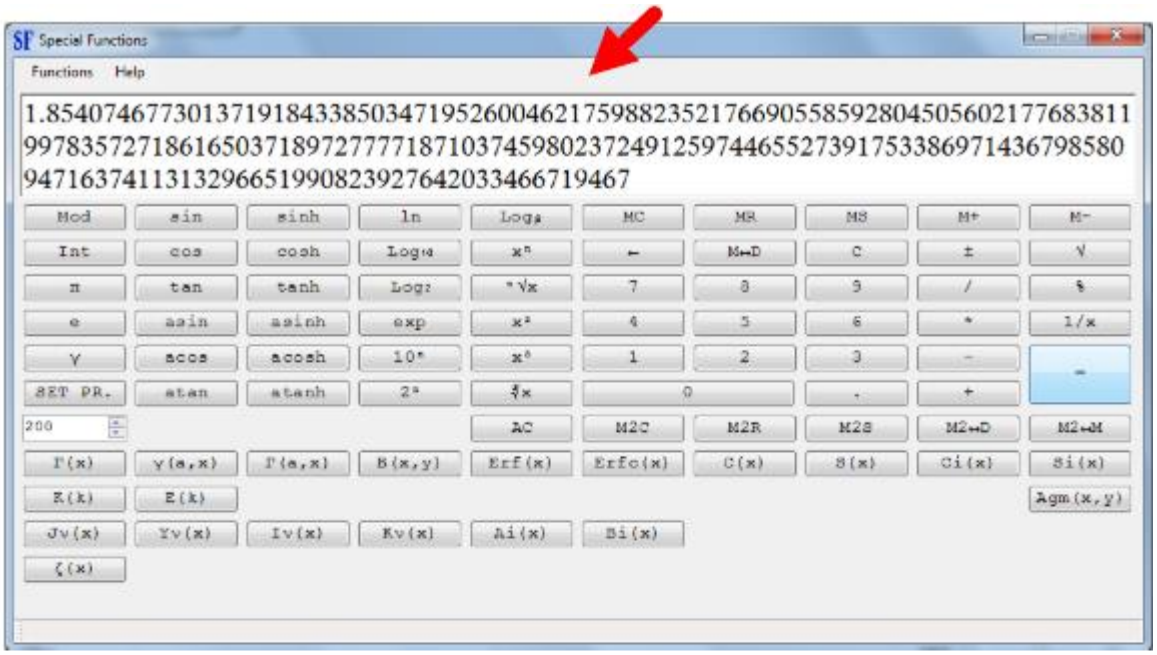
Annotation Abstract: The paper describes the methods and algorithms used for realization of special functions computing with arbitrary precision in the environment of .NET Framework. .NET Framework C# is used as a tool, with the help of the MPIR library. An example is demonstrated, with a program, using the current state of the special functions library realization. Some perspectives for future development are outlined.

Keywords: High precision computation, special functions, computational mathematics.

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, non-linear dynamic systems. But due to their essence, similar calculations are not intended for direct real time control of quickly running technological processes, i. e., widely spread industrial production processes. The present paper describes the first stage of the realization of a system for arbitrary precision computing in the environment of .NET Framework, concerning special functions calculations. The used methods and algorithms for their realization are described The library MPIR [19] is used as a tool. It is a detached version of the arbitrary precision mathematical calculations library, based on GMP (GNU MP – library for arbitrary precision mathematics of GNU). X-MPIR ensures interface in

.NET 33 Framework C# to a library of previously compiled functions of MPIR, realized in C. To our knowledge the latest overview of the special functions calculation is given in Chapter 4 of [1]. In monograph [2] there is a detailed list of sources, which may be used for contingent check up. It is to be noted that intentionally preference is given to the methods, using quickly convergent series in the present realization of the special function library, wherever this is possible and/or practical. In most cases alterative methods exist such as continued fractions, integral presentation, using of iterative relations, and so on. But for computing with arbitrary precision the possibility for simple error evaluation is crucial. The approach adopted is not free of its specific requirements, for instance ensuring consistency of the main series and asymptotic presentation. About the specificity of the asymptotic presentation with divergent series see also [3]. Nonetheless this approach seems to be simpler in the arbitrary precision context, which is anyway apriori specified and should be easily calculated.

Realization and methods used: The realization of the functions, at present, is for a real argument. A basic version of a program-calculator is created implementing immediate usage of the library for the purpose of testing the functions. It is described in details in section 3 "Testing calculator". The generally accessible sources used for the methods realized are [1- 6]. Reference [4] is most intensively used although even there the references "Methods of computation" are not always sufficient, but this does not belittle the exclusive value of the writing for any calculator. Sites [20-22] provide good initial references. Possible sources about specific special functions and constants, as well as respective computational methods when they are non-standard, e.g. power series and asymptotic presentations, are given in place in this section. Many of the functions in the library are internally used to express other functions, e.g. in a respective range or an index type, and some of them, as well as calculating the numerator and denominator in the Bernoulli numbers, are not still represented in the calculator. The special cases of the hyper geometric function are a typical example: ( ) ( ) ( ) ( ) ( ) ! ;,,;,, 0 1 1 1 1 n z bb aa zbbaaF n n n n q n n p qp p q ∑ ∞ = = K K L K . Only the special cases are used in the representations below, where p = 0, q = 1, and p = 1, q = 1, at which (p < q + 1) series is everywhere convergent, if b = b1 is not a negative integer.



```
1.854074677301371918433850347195260046217598823521766905585928045056021776838119
9783572718616503718972777718710374598023724912597446552739175338697143679858094
71637411313296651990823927642033466719467
```

Fig-1

The testing process, as it is well known, is the most time and labor-consuming operation for a computer program. This of course is valid to even greater degree when calculating special functions. It includes several stages and is not still completed. The main checks are for previously known values of a given function with exactly defined argument/s. An even better test is the possibility of comparing a given result when it could be expressed by different functions, thus using different methods. For checking with big arguments values where asymptotic presentations are used, this is crucial. Adding more functions assists this type of tests. There are in the net a lot of on line calculators for different types of special functions which are convenient for initial adjustment. But most of them are of limited precision. A natural opportunity is the parallel using of another program, using e.g. mpmath in Python. It is anyway convenient to have an interactive program for testing. For this purpose a prototype of a calculator is realized. It allows dynamic change of the precision used.

The display field adds automatically a vertical slider for scrolling, if needed, when required by the current computing precision. The format of the numbers displayed is automatically changed depending whether the result needs exponential format. Appropriate rounding is carried out. Indication is available for a pending argument for functions with more than one argument. Digits, decimal point and arithmetic operations may be also entered from the keyboard. There is not yet overflow check.

The problem is specific for the MPIR library also where the floating point numbers exponent is fixed. For 32-bit systems from 68 719 476 768 to 68 719 476 736, depending on the machine word but not equal to it. For a 64-bit system, for which the present special functions library is being developed this range is larger and this brought to underestimation of the problem initially. This overflow check will be probably executed at the level of functions. Some change will be probably necessary at that level of the special functions reaction with regard to non valid argument. In this realization when the argument is non valid they issue a message and return the input and do not cause an exception just for convenience at testing in interactive environment. The adequate approach for the function behavior in an independent environment is to be considered. There the responsibility of entering an admissible argument does not lie on the calling program. The alteration of the current model will require additional efforts for the program-user and namely processing of specific exceptions. The calculator for the moment is in a form that allows easy adding of new functions. The project presumes the 'calculator' to be transformed in a source of references for the used special functions and graphical representations An example follows with a calculator with 200 digits precision.

The first one is faster under the condition that the specified precision is not changed in the frame of the given calculation or the constants are computed with sufficiently high precision, which would not be overridden at computation. The second method is used for the moment After accurate realization of elementary functions constantsπ , e , ln2, ln10, log2, log e, roots and powers of integers and their various combinations including arithmetic operations are available. The constant with the current precision of computing is returned at calling the respective functions. The algorithm proposed by B r e n t and M c M i l l a n in [7] is used for Euler’s constant γ . Brent and McMillan algorithm: Suppose we want to compute Euler’s constant γ with precision up to d decimal digits. If we choose n to be the greatest integer which is less than c + (1/4) ln10d with an appropriate constant c, then where

U(n) and V(n) are computed as follows: Define $\left|\gamma - \frac{U(n)}{V(n)}\right| < \pi e^{4-4c} 10^{-d}$,

Fig-2

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

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

Fig-3

Where Hk is the k-th harmonic number (1 + 1/2 + … + 1/k). Then A0 = – ln n, B0 = 1, U0 = A0, V0 = 1 and for k = 1, 2, …, we receive

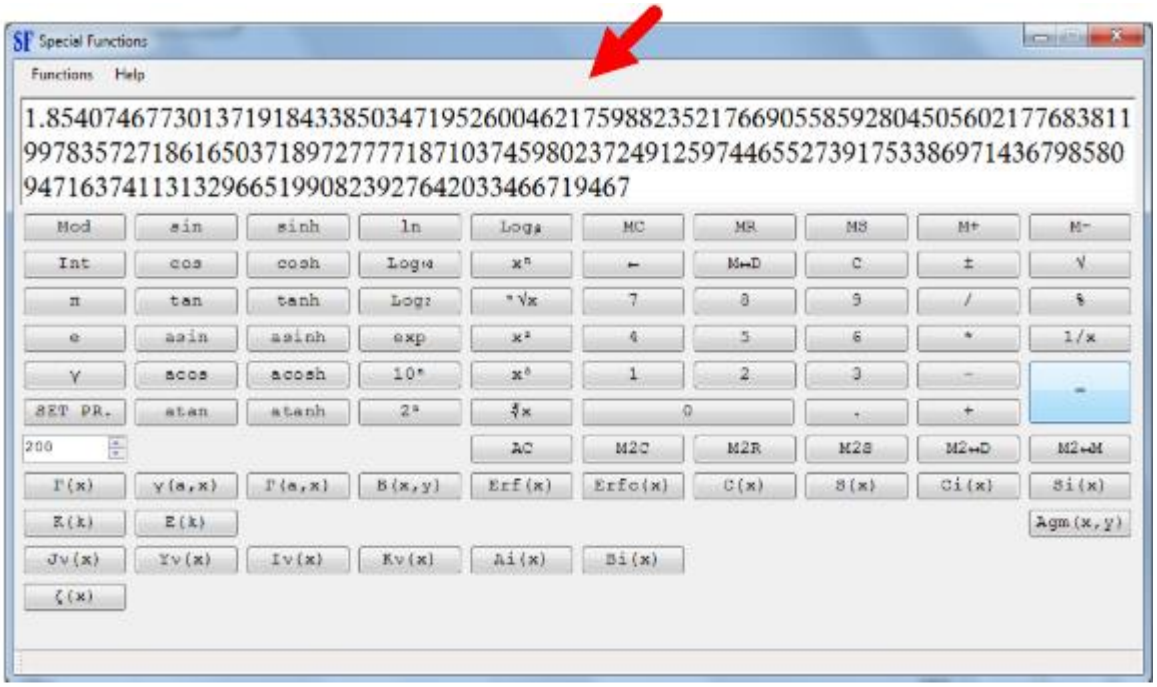
$$B_k = B_{k-1}\frac{n^2}{k^2}, \qquad A_k = \frac{\left(A_{k-1}\dfrac{n^2}{k} + B_k\right)}{k},$$

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

Fig-4

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

Fig-5

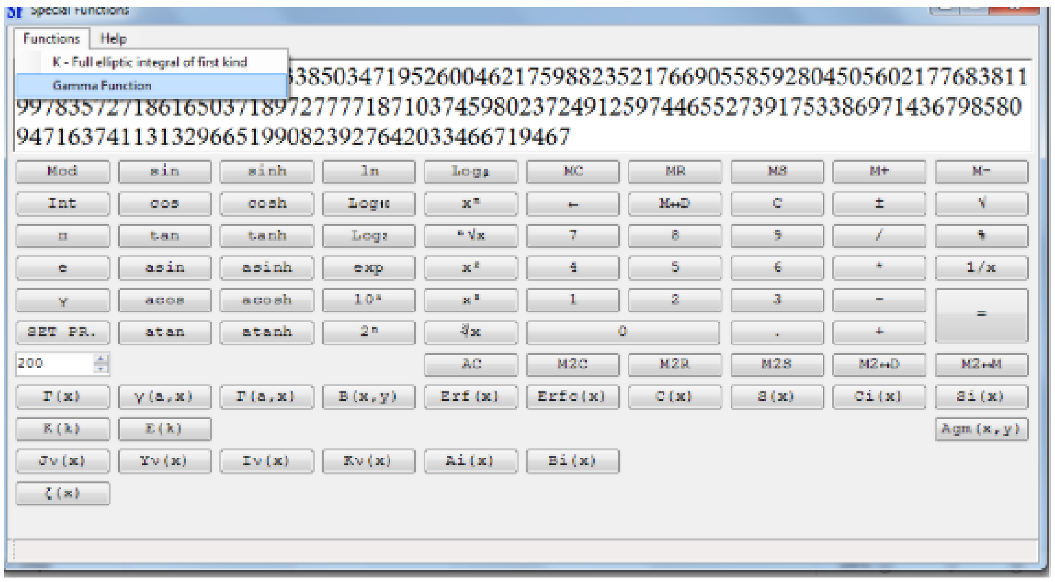


```
1.85407467730137191843385034719526004621759882352176690558592804505602177683811
99783572718616503718972777718710374598023724912597446552739175338697143679858O
94716374113132966519908239276420334667l9467
```

Fig-6

It may be stored by MS and after entering 2, , 1/x, K(x), the left one is received. No illustration is given since the result is the same. Additional facilities are added for easy testing when the equivalent expression is more complicated: storing in a second register (M2S) and M , and ↔ D M , which changes the places of the last received and last stored in the 2 ↔ D respective auxiliary register results.

Consecutive entering: 4, 1/x, Γ(x), 2 x , MS, π , , 1/x, /, 4, =, *, MR, =, yields the right hand side of the equation (Fig. 6).



```
385034719526004621759882352176690558592804505602177683811
99783572718616503718972777718710374598023724912597446552739175338697143679858 0
9471637411313296651990823927642033466719467
```

Fig-7

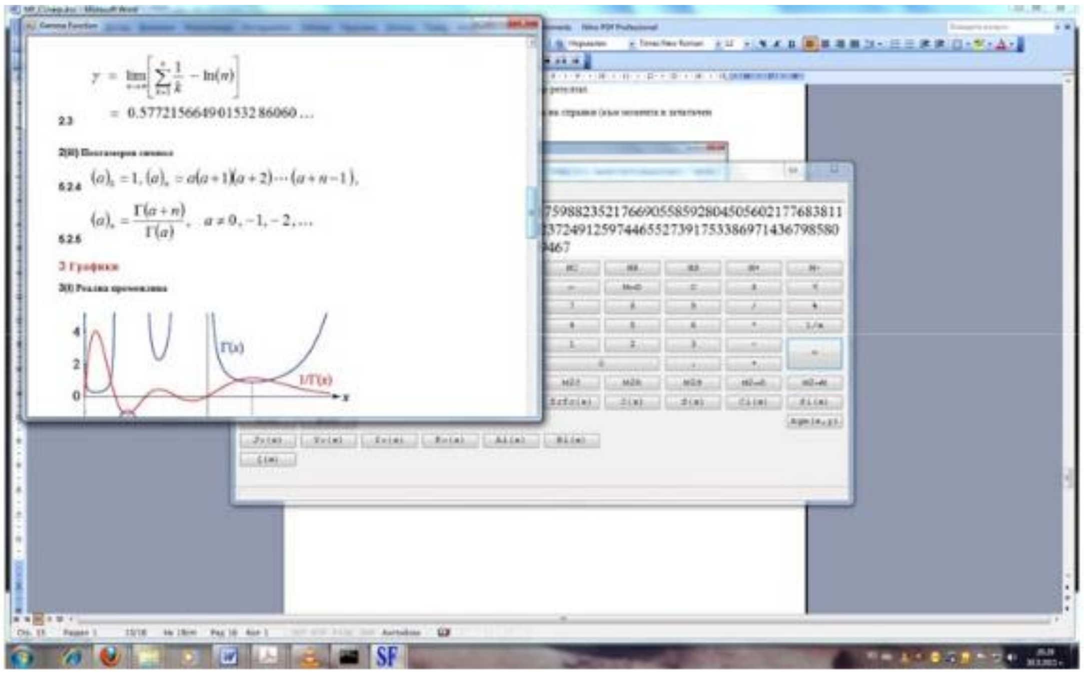


Fig-7

Conclusion: The main problem in the realization of a similar project is in its range – a highly labor-consuming job. As mentioned in the introduction the purposes are in essence carried out to a great degree in other specialized products or packages in other programming languages environment. Leaving away the issue that the greater part of them are paid commercial products, and those which are not require more programming by the user, e.g. for the graphical interface, we reckon that the possibility of arbitrary precision computations combined with the other exclusively rich features of .NET Framework is worth the efforts. The completed environment for arbitrary precision computations, including also methods of the numerical calculus, provides the base for an easy graphical interface which is the next logical step of the project. Probably new types will appear in the future in .NET Framework as continuation of BigInteger but this does not seem happen soon. A calculator of arbitrary precision can by found in web with a limited set of functions, which is based on BigInteger only. This approach is not a good perspective for the present project. The type BigInteger is not designed for this purpose and real numbers calculations based on it is not efficient enough. A more straightforward approach is writing an own class representing floating point numbers with arbitrary precision. But this moves the aims a step aback without mentioning the technical difficulties - writing in C style with mandatory modifiers ‘unsafe’ that returns to the beginning. So the main problem remains - time and labor consumption. So for the full realization of the project besides adherence to the theme and some financial support confidence in its utility and value is necessary.

References

1. Toxir Turg'un o'gli, A. (2023). C# DA FAYLAR ORQALI MA’LUMOTLARNI SARALASH VA FAYLLAR BILAN ISHLASH ALGORITMLARINING XUSUSIYATLARI. In Uz-Conferences (Vol. 1, No. 1, pp. 139-141).
2. Toxir Turg'un o'gli, A. (2023). O ‘RINLARNI ALMASHTIRISH USULLARI. In Uz-Conferences (Vol. 1, No. 1, pp. 133-138).
3. Toxir, A., & Lobar, A. (2023). MOBIL ILOVALAR ORQALI YOSH BOLALARDA UCHRAYDIGAN NUTQ BUZILISHLARINI BARTARAF ETISH. In Uz-Conferences (Vol. 1, No. 1, pp. 906-911).
4. Toxir, A., & Lobar, A. (2023). MOBIL ILOVALAR ORQALI YOSH BOLALARDA UCHRAYDIGAN NUTQ BUZILISHLARINI BARTARAF ETISH. In Uz-Conferences (Vol. 1, No. 1, pp. 906-911).

5. Norqo'ziyev , Q. (2023). MOBIL ROBOTLAR UCHUN YO'LNI REJALASHTIRISH ALGORITMI. *Research and Implementation*. извлечено от https://fer-teach.uz/index.php/rai/article/view/746

6. Норкозиев, К., & Тоджиев, А. (2023). Использование искусственных нейронных сетей при разработке алгоритма поиска оптимального пути мобильных роботов в динамических средах. Информатика и инженерные технологии, 1(1), 25–29. извлечено от https://inlibrary.uz/index.php/computer-engineering/article/view/25025

7. Тоджиев, А., & Норкузиев, К. (2023). The role of artificial intelligence technology in individualized teaching . Информатика и инженерные технологии, 1(2), 153–156. извлечено от https://inlibrary.uz/index.php/computer-engineering/article/view/25014

8. Искандарова З., Иргашева У. Цифровизация и образование для устойчивого развития //Информатика и инженерные технологии. – 2023. – Т. 1. – №. 1. – С. 197-200.

9. Рустамов М. Ж., кизи Иргашева У. А. ЗАДАЧА ВОССТАНОВЛЕНИЯ СКОРОСТЬ ИЗМЕНЕНИЕ ТЕМПЕРАТУРЫ ПО КОСВЕННЫМ НАБЛЮДЕНИЯМ //RESEARCH AND EDUCATION. – 2022. – Т. 1. – №. 2. – С. 22-29.

10. Oybek Kayumov, Obid Kayumov, & Isoqov Adham, Meliyeva Mohira. (2023). ELECTRONIC PLATFORM FOR RECOGNITION AND TEACHING OF SIGN LANGUAGE PICTURES BASED ON UZBEK GRAMMAR. International Journal of Contemporary Scientific and Technical Research, 263–268. Retrieved from https://journal.jbnuu.uz/index.php/ijcstr/article/view/587