2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing

Software Implementation of Numerical Algorithms in Arbitrary Precision

Zinovi L. Krougly
Department of Applied Mathematics and Statistical & Actuarial Sciences
Western University
London, Ontario, Canada N6A5B7
Email: zkrougly@stats.uwo.ca

David J. Jeffrey
Department of Applied Mathematics
Western University
London, Ontario, Canada N6A5B7
Email: djeffrey@uwo.ca

Dina Tsarapkina
Department of Applied Mathematics
Western University
London, Ontario, Canada N6A5B7
Email: dtsarapk@uwo.ca

***Abstract*—We introduce a Matlab *mprec* arbitrary precision library with applications to numerical analysis. For maximum efficiency arithmetic operators and algebraic functions are implemented in the *mpreal* class. The examples are chosen to reflect the diversity of types of problems for which multiple precision can play a useful role.**

I. Introduction

The numerical algorithms in arbitrary precision arises in many applications of science and engineering. We have written an interface code, defined Matlab classes, and linked them to a compiled form of the ARPREC C++ source code [1].

For computational efficiency most of the basic arbitrary precision (AP) Matlab functions use the interface to C++. In addition we incorporate algorithms for ordinary differential equations, and for some advance functions such as Newton's iterations, Lambert W function, and numerical inversion of the Laplace transform that provides higher accuracy and performance. The accuracy of AP calculations can be changed by initializing certain algorithmic parameters.

II. Implementation of the *mprec* package

The *mprec* package includes the following basic operations and functions [2]:

- Arithmetic operations, common transcendental functions, including cos, sin, tan, arccos, arcsin, arctan, exp, ln, log, Erf, gamma and other functions
- Supports AP datatypes: mp_real, mp_int and $mp\_complex$, vectors and matrix calculations
- Definitions of useful mathematical constants such as $e, \log_2 e, \log_{10} e, \ln 2, \ln 10, \pi, \pi/2, \pi/4, 1/\pi, 2/\pi, 2/\sqrt{\pi}, \sqrt{2}, 1/\sqrt{2}$
- Matlab Code *mpreal* class to incorporate AP operations and functions in a natural mathematical expressions
- C++ code for AP computing, high performance and full portability
- Common numerical algorithms (solving complex equations, Newton's iterations, ordinary differential equations, etc)
- Special routines for Matlab interface with C++: to compute fundamental mathematical constant and special values such as $\pi, e, \ln 2, epsilon$; utilize and execute string containing Matlab expressions, and convert C++ output arguments from the expressions into strings

The *mprec* multiple precision function library is a collection of Matlab m-files for AP floating point calculations. In general, it is possible to write m-files for each arithmetic and algebraic operations. But this requires having to rewrite each software application which introduces significant debugging difficulties. Another important aspect to consider is the possibility of rewriting some of the fundamental applications with high performance capabilities by making only minor changes. In this regard, a more efficient approach, is to incorporate these conversions for mathematical operations as in mathematical convention, using the Matlab operator overloading features of object-oriented programming.

The *mprec* numbers are typically denoted by x, y, z, and the string variables are denoted by s. The software supports input, of real and complex numbers. Some of the basic operations are as follows.

To create a *mprec* class within the Matlab environment and provide useful functionality the following methods are implemented:

- Overloaded basic arithmetic operations: addition (+), substraction (-), multiplication (*), division (/) and power(^).
- Overloaded mathematical function, such as roots, $\sin, \cos, \exp, \log$ and some advanced functions and fundamental algorithms, as $newton$ (Newton's iterations), $lambertw$ (Lambert W function), $gavste$ (Gaver-Stehfest algorithm for inverse Laplace transform), and $rk4$ (Runge-Kutta method for Ordinary differential equations).
- Overloaded relational operations: equal (==), not equal (!=), less then ($<$), greater then ($>$), less than or equal ($<=$), greater than or equal ($>=$).

See Tab. I for a basic list.

In the next sections we present some applications, where the extended precision allows for construction of efficient

978-1-4799-3035-7/14 $31.00 © 2014 Crown Copyright
DOI 10.1109/SYNASC.2013.25

TABLE I. BASIC ARBITRARY PRECISION ARITHMETIC AND ALGEBRAIC OPERATIONS IMPLEMENTED IN *mpreal* CLASS

No	Function	Operation	Data Type
1.	z = x + y	z = x + y	x, y, z AP complex
2.	z = x - y	z = x - y	x, y, z AP complex
3.	z = x × y	z = x × y	x, y, z AP complex
4.	z = x / y	z = x / y	x, y, z AP complex
5.	z = sin(x)	z = sin(x)	x, z AP real
6.	z = cos(x)	z = cos(x)	x, z AP real
7.	z = asin(x)	z = arcsin(x)	x, z AP real
8.	z = acos(x)	z = arccos(x)	x, z AP real
9.	z = atan(x)	z = arctan(x)	x, z AP real
10	z = exp(x)	$z = e^x$	x, z AP complex
11.	z = log(x)	z = ln(x)	x, z AP real
12	z = sqrt(x)	z = $\sqrt{x}$	x, z AP real
13.	z = x^n	z = x^n	x, z AP real, n integer
14.	z = x^y	z = x^y	x, y, z AP real
15.	z = double(x)	Conversion	x string, z double
16.	z = char(x)	Conversion	x AP, string, z AP complex
17.	z = char(x)	Conversion	x AP, string, z AP complex

TABLE II. TABLE OF LAPLACE AND INVERSE TRANSFORMS FOR TEST FUNCTIONS USED IN NUMERICAL CALCULATIONS

$F_{01}(s) = s^{-1/2}e^{-s^{-1}}$	$f_{01}(t) = (\pi t)^{-1/2}\cos(2t^{1/2})$
$F_{02}(s) = (s+1/2)^{-1}$	$f_{02}(t) = e^{-t/2}$
$F_{03}(s) = ((s+0.2)^2+1)^{-1}$	$f_{03}(t) = e^{-0.2t}\sin(t)$
$F_{04}(s) = s^{-2}$	$f_{04}(t) = t$
$F_{05}(s) = (s^2+1)^{-1}$	$f_{05}(t) = \sin(t)$
$F_{06}(s) = (s^2-1)(s^2+1)^{-2}$	$f_{06}(t) = t\cos(t)$
$F_{07}(s) = (s+1/2)^{1/2} - (s+1/4)^{1/2}$	$f_{07}(t) = (e^{-t/4} - e^{-t/2}) \times (4\pi t^3)^{-1/2}$
$F_{08}(s) = e^{-4s^{1/2}}$	$f_{08}(t) = 2e^{-4/t}(\pi t^3)^{-1/2}$
$F_{09}(s) = \tan^{-1}(s^{-1})$	$f_{09}(t) = t^{-1}\sin(t)$
$F_{10}(s) = \frac{e^{-1/s}}{\sqrt{s}}$	$f_{10}(t) = \frac{\cos(2\sqrt{t})}{\sqrt{\pi * t)}}$

algorithms.

III. GAVER-STEHFEST ALGORITHM FOR INVERSE LAPLACE TRANSFORM

The numerical inversion of Laplace transform arises in many areas of science and engineering. Stehfest [3] derived the Gaver-Stehfest algorithm for the numerical inversion of Laplace transforms. For most of the more interesting problems, however, numerical inverting often has numerical accuracy problems [4], [5], [6]). As such, small rounding errors in computation may significantly offset the results, rendering these algorithms impractical to apply. With extended precision, we are able to add additional significant figures, and in doing so produce results that converge to the solution without falling victim to the serve truncation that may happen in standard double arithmetic.

This demonstration applies the algorithm to determine the inverse Laplace transforms of ten test functions to various type numerical accuracy. The inverse functions and corresponding test functions are presented in Tab. II. The first nine function are taken from [4], and the last one from [5].

The Gaver-Stehfest method uses the summation:

$$f(t) \approx \ln 2/t \sum_{n=1}^{N} K_n F(n \ln 2/t). \quad (1)$$

The K_n coefficients only depend on the number of expansion terms, N (which must be even), they are:

TABLE III. CALCULATION ERRORS OF THE INVERSE LAPLACE TRANSFORM IN DOUBLE PRECISION FOR THE FUNCTIONS IN TAB. II. VALUES 1.7E-4 $\equiv 1.7 \times 10^{-4}$

	N = 8	N = 16	N = 18	N = 32
F_{01}	1.2E+0	1.7E-4	3.6E-4	8.0E+5
F_{02}	1.7E-1	2.0E-4	1.6E-4	2.6E+5
F_{03}	1.7E+1	2.2E+0	1.1E+0	5.1E+4
F_{04}	9.3E-2	2.6E-5	8.4E-5	7.1E+4
F_{05}	5.1E+1	1.1E+1	5.9E+0	6.2E+4
F_{06}	2.6E+2	8.1E+1	6.4E+1	5.4E+4
F_{07}	4.4E-3	5.7E-4	1.1E-2	2.1E+7
F_{08}	9.7E-2	1.8E-3	6.6E-4	1.1E+3
F_{09}	8.9E+0	1.2E+0	6.2E-1	2.9E+5
F_{10}	1.9E+0	1.7E-4	3.8E-4	7.6E+5

$$K_n = \sum_{k=[(n+1)/2]}^{min(n,N/2)} \frac{(-1)^{n+N/2} k^{N/2}(2k)!}{(\frac{N}{2}-k)!k!(k-1)!(n-k)!(2k-n)!}. \quad (2)$$

For each function we calculate an error, E, defined by

$$E = (\sum_{i=1}^{30}(f(i/2) - f_\alpha(i/2))^2/30)^{1/2}, \quad (3)$$

as the measure for the accuracy of the numerical solution [4]. Let $f(t)$ be the analytical solution. We denote by $f\alpha(t)$ the numerical estimate of the exact solution. Then E gives the root-mean-square deviation between the analytical and numerical solutions for the t values 0.5, 1, 1.5, ..., 15.

The K_n coefficients become very lalrge and alternate in sign for increasing n. The sum (1) begins to suffer from cancelation for large $N \geq$ the number of decimal digits of precision (e.g., double precision = 16).

1) Testing Gaver-Stehfest inversion algorithms in double precision: In Tab. III we report some numerical results related to the test functions presented in Tab. II.

All the calculations in Tab. III were done in double precision. In Tab. II we show a number of functions and their exact inverse functions. The numerical solutions for the functions in Tab. II were compared to 200 points along the exact solution, and their norms were added. Table III shows the sums of these errors which gives a numerical representation of how good a solution truly is.

When looking at numerical inversion, it is important to note the accuracy with a varying number of terms and precision. We compare the inverses using Matlab Gaver-Stehfest implementation and observe the accuracy of the inversions as we increase the number of the expansion terms and precision. However, there exists a limit to adding additional terms. As we increase the number of terms, N, in the computation we quickly discover that the numerical inversion becomes unstable and our function is dominated by numerical error. For double precision it is reasonable to only allow a number $N = 18$. In many cases 18 expansion terms is sufficient to approximate a function almost exactly, however there are times where a noticeable difference is seen. We have provided a number of trial functions whose exact solutions are known, to identify the ones that would benefit from the use of extended precision. Using extended precision allows to combat the

numerical limitation that we experience when dealing with double precision. Thus, we can use a larger number of terms for those functions without encountering noise.

The calculations in double precision for all functions and any number of terms are at most roughly 10^{-5}. The accuracy is very poor (the functions F_{03}, F_{05}, F_{06} and F_{09}). The convergence is poor for all these functions for the number of terms N = 8, 16, 18, and even worse if the number of terms $N \geq 32$. If we continue increase the number of terms, i.e. for $N = 128$, we get NaN indicating Not-a-Number, that the Gaver-Stehfest algorithm in double precision was unable to provide even an order of magnitude estimates.

The results shown in Fig. 1 - Fig. 10 correspond to the function $F_{01} - F_{10}$ in Tab. II. The inverse Laplace transforms are given for ten functions evaluated in double precision. The curves illustrate the solutions for different number of expansion terms in comparison to the exact solution. The term "exact solution" in the figures means the numerical estimate of the exact solution.

In figures Fig. 2, Fig. 4 and Fig. 7 even with only $N = 8$ expansion terms we are able to converge to the solutions which are indistinguishable to the eye.

The Fig. 1, Fig. 8 and Fig. 10 provide the solutions that are indistinguishable to the eye between $N = 16$ and $N = 18$. The value N is chosen to provide the closest solution to the exact value. Due to the functions being slightly different, the error may begin to worsen the solution at an earlier term (N=18), or the additional terms may continue to benefit the solution even with 16 digits.

In the remaining figures, Fig. 3, Fig. 5 Fig. 6 and Fig. 9, we can see the difference between the best approximation and the exact solution. For these problems, arbitrary precision is of interest. We are able to add additional terms to improve the solution, and delay the onset of the noise.

In all the figures, when looking at the plot using $N = 32$ we see that using too many terms causes rounding error to overtake the numerical solution, and we essentially obtain noise. This is because as small changes in large numbers are manipulated, we experience severe truncation of the changes which lead to unstable solutions.

2) Implementation of Gaver-Stehfest algorithm in arbitrary precision: The Gaver-Stehfest algorithm was implemented in arbitrary precision, taking advantage of the arbitrary number of significant digits. Below is a demonstration of how the package can be used to compute an inverse Laplace transform on a function predefined in our code. The parameter *fun* represents the name of the function to be transformed, i.e. $fun = 1$ corresponds to the function F_{01} in Tab.II. The time at which the inverse Laplace transform of the original function is required is denoted by *t*. *N* is the number of expansion terms to be used in the computation, which must be an even number, and *prec* is the precision to be used. The path_ reflects the path to the C++ $mpcalc$ dll project, where Gaver-Stehfest algorithm (function $gavste$) employed. Upon running this test function script, we will obtain in the output the value of the inverse Laplace transform, *invlt* and the *err* corresponds to formula (3) when this result is compared to the numerical estimate of the exact value.

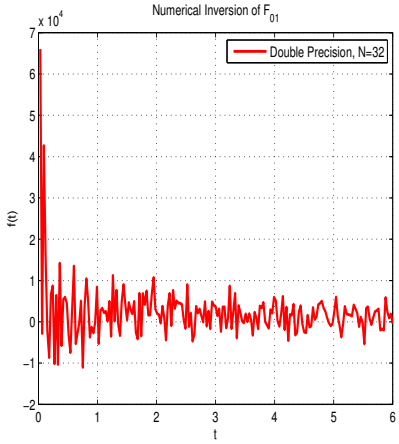

Fig. 1. Inverse Laplace transform of the function F_{01} evaluated in double precision

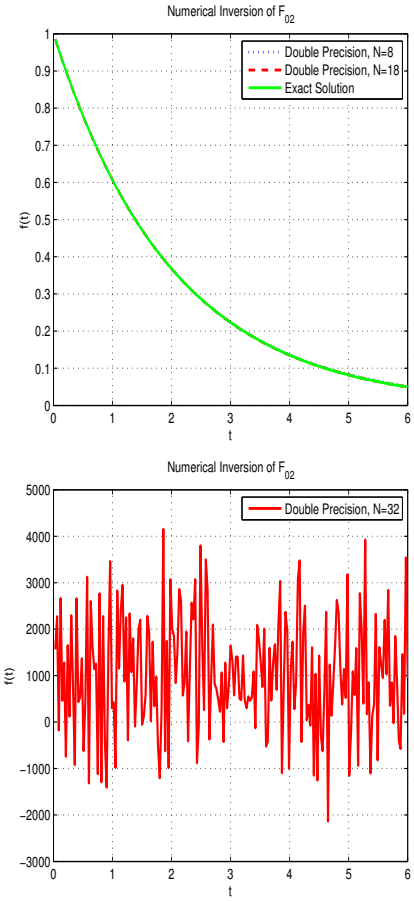

Fig. 2. Inverse Laplace transform of the function F_{02} evaluated in double precision

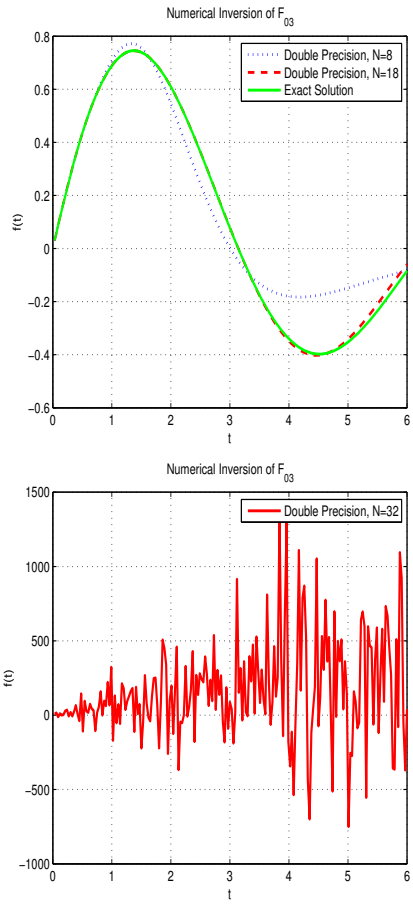

Fig. 3. Inverse Laplace transform of the function F_{03} evaluated in double precision

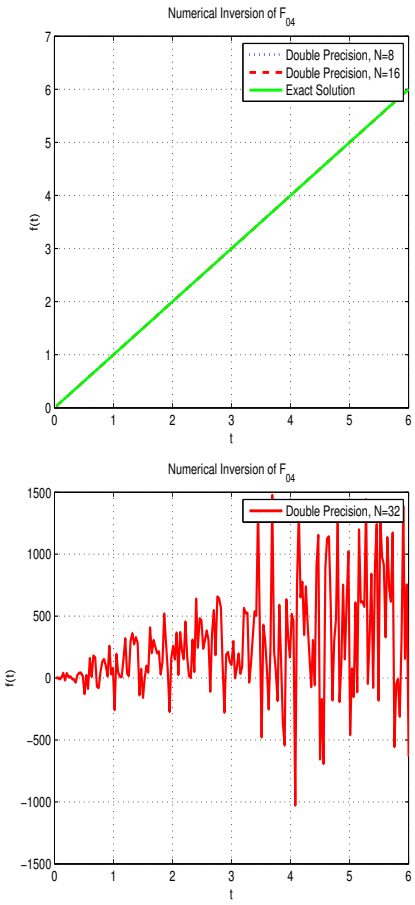

Fig. 4. Inverse Laplace transform of the function F_{04} evaluated in double precision

The test function can be run with the code below

```
% Numerical inversion of the Laplace transform using Gaver-Stehfest
% algorithm with extended precision calculations
function [inv_lt, err] = gavste_test(fun,t,L, prec)
%% Input
%    fun       The name of the function to be inverse transformed
%              (the number corresponds to test function in mpcalc
%              dll project)
%    t         The value of t where the inverse Laplace transform
%              to the (unknown)
%              original function f(t)is required (t > 0), usually
%              a snapshot of time(examples: t = 0.5, 1, 1.5, ...,
%              15, so on)
%    N         The number of expansion terms, which must be even
%              (examples: N = 4, 16, 32, 64, 128, 256, so on)
%    prec      The number of digits in arbitrary precision
%              calculations used (examples: prec = 16, 32, 100,
%              ..., 1000)
%% Output
%    invlt     The value of the inverse laplace transform
%    err       Error of the inverse Laplace transform
%
%              Example: [inv_lt, err] = test_gavste(1,2,32,100)

global path_ precision;
mputil;
precision = prec;
fn = '2';  % Inverse calculation
k = fun;
kp = mpreal([k,L]);
tn = mpreal(t);
tn.imag = 100;
responce = gavste(kp,tn,fn);
inv_lt = responce.real;
fn = '3';  % Error calculation
responce = gavste(kp,tn,fn);
err = responce.real;
end
```

The output is:

TABLE IV. CALCULATION ERRORS OF THE INVERSE LAPLACE TRANSFORM IN ARBITRARY PRECISION (100 DIGITS) FOR THE FUNCTIONS IN TAB. II. VALUES 9.6E-2 $\equiv 9.6 \times 10^{-2}$

	N = 4	N = 16	N = 32	N = 64	N = 128	N = 256
F_{01}	9.60E-2	1.6E-4	8.1E-12	1.8E-26	2.3E-33	6.3E+53
F_{02}	1.5E-2	6.4E-6	5.2E-11	1.3E-23	3.4E-34	6.1E+52
F_{03}	1.7E-1	5.0E-2	4.4E-3	4.4E-7	1.5E-21	1.0E+52
F_{04}	3.4E-1	3.9E-7	1.5E-14	3.1E-29	1.4E-34	1.1E+52
F_{05}	6.6E-1	4.7E-1	1.4E-1	1.3E-4	1.0E-17	1.1E+52
F_{06}	6.0E+0	5.2E+0	1.9E+0	7.0E-3	5.6E-16	1.1E+56
F_{07}	6.3E-4	6.4E-8	1.9E-13	2.7E-26	1.3E-31	7.2E+55
F_{08}	3.6E-3	6.8E-6	2.3E-9	1.9E-16	8.6E-31	2.5E+52
F_{09}	1.1E-1	4.1E-2	7.5E-3	4.9E-6	8.6E-20	4.6E+59
F_{10}	9.6E-2	1.6E-4	8.1E-12	2.0E-26	2.3E-33	6.3+E53

```
invlt = 10 ^ -1 x -3.795389758243676381129800678241861148...
840967617592231946344278938954985244981787821768215557460943428

err = 10 ^ -12 x 8.13762246217538094824142851779367405194 5...
118475730848296549470231938397691733197561032902744637450831
```

Next we report some numerical results in AP to the test functions presented in Tab. II. The calculations are given in Tab. IV for 100 digits and Tab. V for 1000 digits.

For the number of terms preceding 100, the accuracy is poor (particularly in the functions F_{05} and F_{06}) and improves

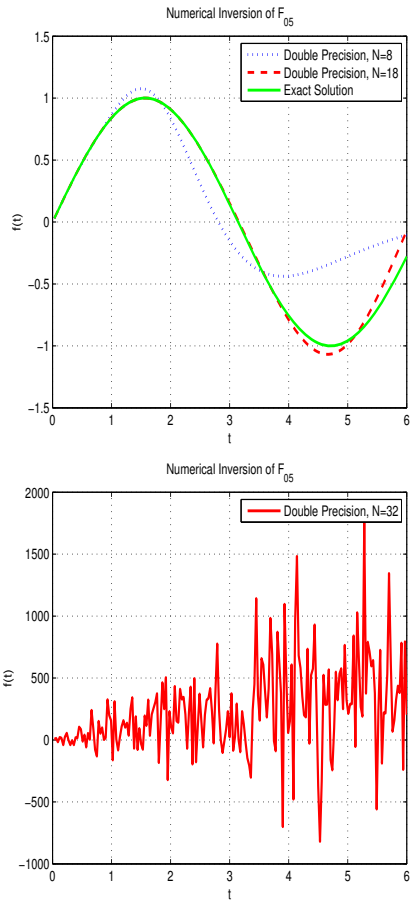

Fig. 5. Inverse Laplace transform of the function F_{05} evaluated in double precision

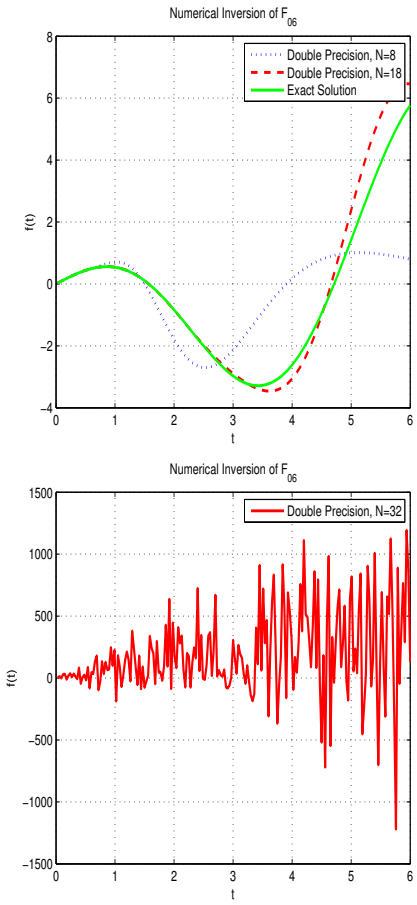

Fig. 6. Inverse Laplace transform of the function F_{06} evaluated in double precision

TABLE V. CALCULATION ERRORS OF THE INVERSE LAPLACE TRANSFORM IN ARBITRARY PRECISION (1000 DIGITS) FOR THE FUNCTIONS IN TAB. II. VALUES 9.6E-2 $\equiv 9.6 \times 10^{-2}$

	N = 16	N = 32	N = 64	N = 128	N = 256	N = 512
F_{01}	1.6E-4	8.1E-12	2.0E-26	1.3E-55	7.6E-114	4.0E-230
F_{02}	6.4E-6	5.2E-11	1.3E-23	5.0E-55	5.1E-119	2.8E-235
F_{03}	5.0E-2	4.4E-3	4.4E-7	1.5E-21	6.8E-66	6.2E-178
F_{04}	3.9E-7	1.5E-14	3.1E-29	2.0E-58	1.2E-116	6.4E-233
F_{05}	4.7E-1	1.4E-1	1.3E-4	1.0E-17	2.0E-60	1.9E-175
F_{06}	5.2E+0	1.9E+0	7.0E-3	5.6E-16	3.0E-58	2.6E-173
F_{07}	6.4E-8	1.9E-13	2.7E-26	1.3E-58	7.6E-120	4.3E-236
F_{08}	6.8E-6	2.3E-9	1.9E-16	8.7E-31	1.9E-59	1.4E-120
F_{09}	4.1E-2	7.5E-3	4.9E-6	8.6E-20	1.9E-62	9.6E-179
F_{10}	1.6E-4	8.1E-12	2.0E-26	1.3E-55	7.6E-114	4.0E-230

to at most roughly 10^{-17} if we increase the number of terms to 128. Further increase the number of terms to 256 doesn't provide convergence due to roundoff errors.

For the number of presiding 1000, the accuracy is very good for all functions and is at least roughly 10^{-16}, 10^{-58} and 10^{-120} for the number of terms accordingly N = 128, 256, and 512.

The MATLAB functions can be evaluated using either command syntax or function syntax. The code inside Matlab functions above will be executed when you call the function. The functions are flexible, you can call them using different arguments. See comments inside the functions for explanation.

Many test examples illustrate different aspects of arbitrary precision numerical techniques.

IV. CONCLUSION

We have introduced the arbitrary precision library, as a collection of Matlab m-files. With the assistance of the implemented *mpreal* class, we presented the arithmetic and algebraic operations as in mathematical convention. For maximum efficiency, many of the functions are implemented by interfacing Matlab to C++.

The list of classical algorithms are implemented, such as Newton's iterations, Lambert W function, and Runge-Kutta algorithm for ordinary differential equations.

We discussed in detail the implementation of the Gaver-Stehfest algorithm for the Laplace transform inversion allowing the advantage of using an arbitrary number of significant digits. We observe the accuracy of the inversions as we increase the number of the expansion terms and precision, which ultimately leads to stable solutions.

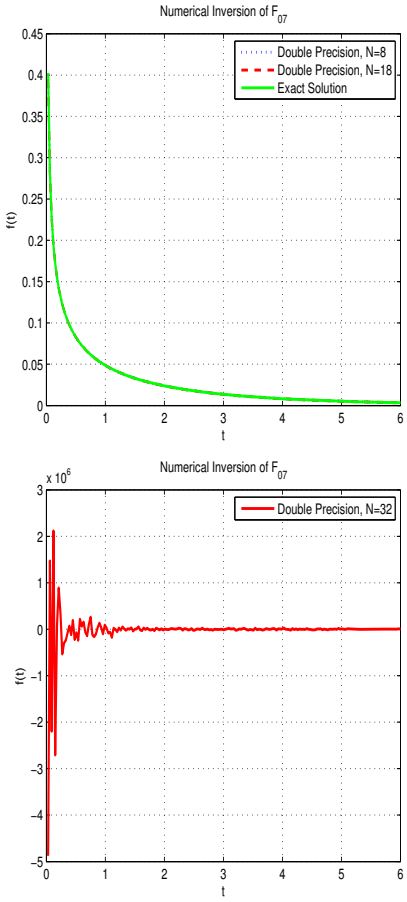

Fig. 7. Inverse Laplace transform of the function F_{07} evaluated in double precision

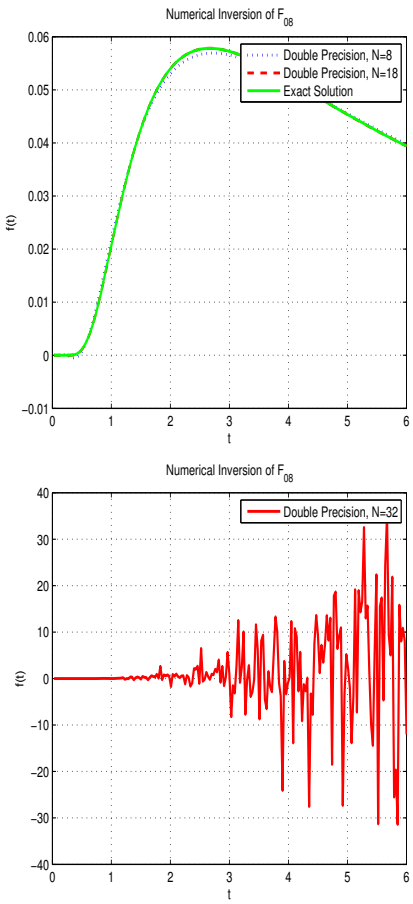

Fig. 8. Inverse Laplace transform of the function F_{08} evaluated in double precision

REFERENCES

[1] D. H. Bailey, Y. Hida, X. L. Li, B. Thompson, *ARPREC: An Arbitrary Precision Computation Package*, http://crd.lbl.gov/ dhbailey/dhbpapers/arprec.pdf., 2002.

[2] Z. L. Krougly, D. J. Jeffrey, *Implementation and Application of Extended Precision in Matlab*, in: N. Mastorakis et al (Eds.), Proceedings of the Applied Computing Conference ACC'09, WSEAS Press, 2009: 103-108.

[3] H. Stehfest, *Algorithm 368: Numerical Inversion of Laplace Transform*, Communications of the ACM, 13(1), 1970: 4749.

[4] B. Davies, B. Martin, *Numerical Inversion of the Laplace Transform, A Survey and Comparison of Methods*, Journal of Computational Physics 33(1), 1979: 132.

[5] A. Murli, M. Rizzardi, *Algorithm 682 Talbots Method for the Laplace Inversion Problem*, ACM Transactions on Mathematical Software 16(2), 1990: 158-168.

[6] K. L. Kuhlman, *Review of Inverse Laplace Transform Algorithms for Laplace-Space Numerical Approaches*, Numerical algorithms 63(2), 2013: 339-355.

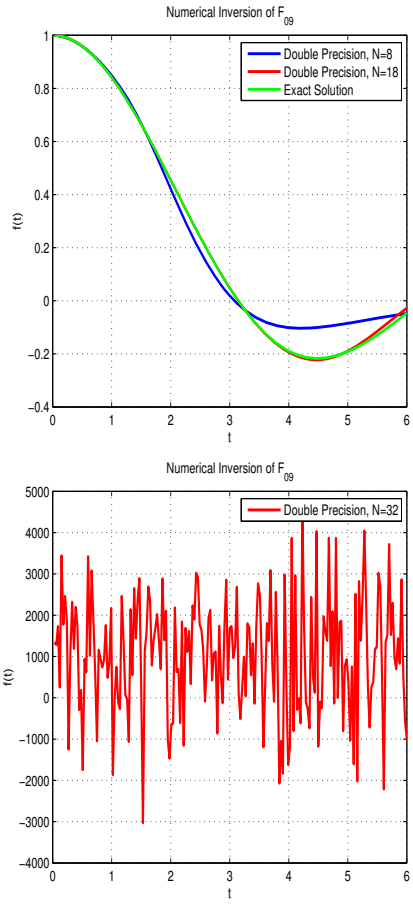

Fig. 9. Inverse Laplace transform of the function F_{09} evaluated in double precision

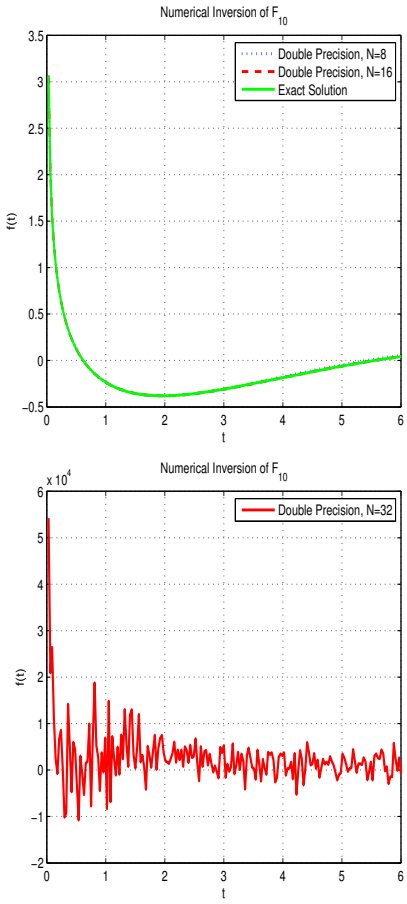

Fig. 10. Inverse Laplace transform of the function F_{10} evaluated in double precision