Implementation and application of extended precision in Matlab

Z.L. KROUGLY
Department of Statistical & Actuarial Sciences
The University of Western Ontario
London, Ontario, Canada
zkrougly@stats.uwo.ca

D.J. JEFFREY
Applied Mathematics
The University of Western Ontario
London, Ontario, Canada
djeffrey@uwo.ca

Abstract: A multiple precision library for floating-point calculations to any number of digits has been implemented in Matlab. The library is based on the ARPREC library. One application is discussed in detail, namely the evaluation in the complex plane of special functions in regions of bad conditioning. Through the use of Matlab classes, all the basic arithmetic operations are accessible using Matlab syntax, even though the fundamental operations are coded in C++. Arithmetic supports both real and complex data to arbitrary precision. The level of the precision being used can be changed at any time. Many elementary functions are also available in Matlab syntax, although not all of them are offered for complex arguments.

Key–Words: Arbitrary precision, special functions, condition number, Lambert W function

1 Introduction

Many applications in science require high accuracy beyond that which can be obtained using IEEE double precision. Applications which benefit from increased precision include simulations, power series, and solutions of nonlinear and differential equations. Recent applications to natural phenomena are contained in Boychuk *et al.* [4] and Krougly *et al.* [12], where extensive investigations of how the distribution selection for rate of spread and spotting distance will affect the spread of forest fire. Other applications are to sensitivity analysis, and stability and optimization techniques in nonlinear systems, such as closed network models (Krougly and Stanford [13]; Casale and Serazzi [7]).

The development of fundamental algorithms in IEEE double-double and quad-double floating point arithmetic was presented in Dekker (1971), Knuth (1998), Shewchuk (1997). One of the first multiple-precision packages was due to Brent (1978), and developed in Fortran 66. His package includes subroutines for evaluating elementary and special functions in multiple precision floating-point arithmetic. Brent's multiple precision package did not support complex arithmetic. Many basic properties of high-precision computations are implemented in the ARPREC software package (Borwein and Bailey, 2004, Bailey, 2002, 2005). Possibly that is the most comprehensive package in multiple-precision software. ARPREC software includes C++ and Fortran-90 modules, and can be downloaded from [1]. This package provides complex arithmetic and some complex elementary functions. Another well-known library is the GMP library, but this does not support many elementary functions.

For many engineers and mathematicians, MATLAB provides a familiar and efficient problem solving environment. Standard Matlab offers only double precision accuracy, and so for applications such as those described above, it would be beneficial to implement multiple precision facilities.

We have implemented a library for high-precision numerical calculations based on the ARPREC library. We have writing interface code, defined Matlab classes, and linked to a compiled form of the ARPREC source code. Thus, we enable programs written in Matlab syntax to access the ARPREC library and compute at any set precision.

2 Special function application

We now describe a specific motivating application. The evaluation of a real-valued function has been studied in great detail. In particular the conditioning of the problem is well understood. The theory for complex-valued functions is less well developed. We are interested in evaluation of the Lambert W function.

Lambert W function plays an important role in science and mathematics [8]. It is the inverse function of $f(w) = we^w$ where $w \in \mathbb{C}$, and it is multi-valued, being denoted by $W_k(z)$, where k is the branch number. The branches of the Lambert W function can be seen in Fig. 1.

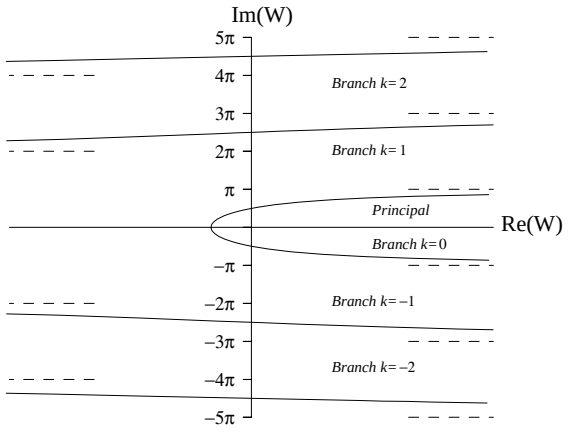

Figure 1: The ranges of the branches of the Lambert W function.

The W function may be evaluated by Newton iteration or by the recurrence formula [8]

$$w_{j+1} = w_j - \frac{w_j e^{w_j} - z}{e^{w_j}(w_j+1) - \frac{(w_j+2)(w_j e^{w_j}-z)}{2w_j+2}} \quad (1)$$

We are particularly interested in evaluation when the argument and branch of W are such that W takes a value near the imaginary axis. Let the imaginary axis be $W = it$ for $-\infty < t < \infty$, then in the z plane, the (pre-)image is $z = te^{i(t+\pi/2)}$. In Fig. 2, the spirals in the z-plane show the locus of points where the condition number for the evaluation of the real part of W is high. This has consequences for accuracy and for the convergence of iterative schemes such as Newton's method, or Halley's method (the scheme given in (1) is a Halley's method). We illustrate the problem with a numerical example. Suppose

$$z = -\pi/2 + 10^{-6} i \, .$$

Let the IEEE double precision approximation to this number be denoted z_{ieee}. Because of conversion to binary data, Matlab will actually work with

$$z_{ieee} = -1.570796326794896 + (9.9999999999999995)10^{-7} i \, .$$

The exact value was calculated using the present software, and we compare that with the value Matlab obtained using IEEE arithmetic.

$$W(z) = (2.884005769\,\mathbf{542417})10^{-7} + 1.5707958737766 87 i$$
$$W(z_{ieee}) = (2.884005769\,\mathbf{265029})10^{-7} + 1.5707958737766 87 i$$

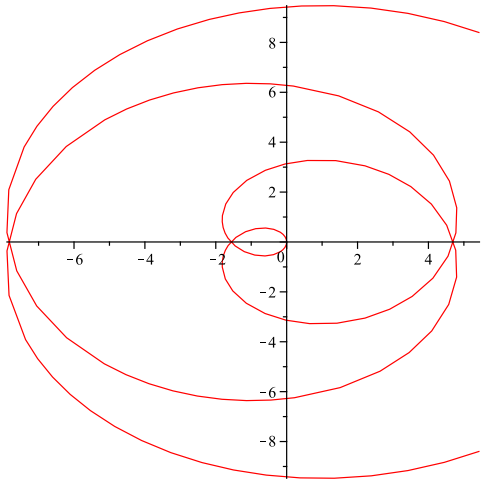

Figure 2: The spirals in the z plane where the condition number of the real part is high.

where the bold digits draw attention to the difference between the two values. Further, Newton iteration does not actually converge fully. The digits shown in bold above will oscillate randomly between iterations. This is shown in the following brief Matlab session.

```
>> N=@(z,w)w-(w*exp(w)-z)/((1+w)*exp(w));
>> z=-pi/2+1e-6*i;
>> w=0.2884e-7+pi*i/2;
>> for k=1:20; w=N(z,w); end;
>> w=N(zd,w);real(w)
ans =
       2.884005768098387e-007
>> w=N(zd,w);real(w)
ans =
       2.884005767927381e-007
>> w=N(zd,w);real(w)
ans =
       2.884005769817498e-007
```

Therefore a relative error of 10^{-10} is the smallest one can hope for using double precision.

The conclusion is that if Matlab is going to deliver values accurate to 0.5 ULP (Units in the Last Place), then computation at higher precision will be needed. This is the reason for starting this project.

3 Implementation of package

The `mprec` package includes the following:

- Compiled C++ ARPREC code (as Matlab MEX file)

- Matlab Code defining class *mpreal* to incorporate AP operations and functions in natural mathematical expressions
- Special routines for C++ interface with Matlab: to utilize and execute strings containing Matlab expressions, and to convert C++ output arguments from expressions into strings.
- Arithmetic operations, common transcendental functions, including cosine, sine, tangent, 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, $\log_2 e$, $\log_{10} e$, $\ln 2$, $\ln 10$, π, $\pi/2$, $\pi/4$, $1/\pi$, $2/\pi$, $2/\sqrt{\pi}$, $\sqrt{2}$, $1/\sqrt{2}$.
- Common numerical algorithms (solving equations over the complex field, Newton iteration, ordinary differential equations, etc)

To create *mprec* class within MATLAB environment and provide useful functionality following methods are defined within the *mprec* class file:

- Overloaded basic arithmetic operations: addition (+), subtraction (-), multiplication (*), division (/) and power(^).
- Overloaded relational operations: equal (==), not equal (!=), less than ($<$), greater than ($>$), less than or equal ($<=$), greater than or equal ($>=$).
- Overloaded Matlab functions, such as `roots`, sin, cos, exp, log and some advance functions and fundamental algorithms, as `newton` for Newton iteration, `lambertw` (Lambert W function) and Runge-Kutta fixed-step method for ODE.

See Table 1, for a full list.

4 Examples and tests of coding

We now give some examples of extended precision calculations using Matlab and the *mprec* package. The *mprec* files must be stored somewhere in the Matlab path, or the path must be extended to include the directory in which the files are stored. These sample codes also test the accuracy of the installation.

4.1 Fixed-point iteration

We begin by solving a fixed-point iteration for a root of a cubic polynomial.

```
% We set desired precision,
% here 40 decimal digits.
precision = 40;

% The constructor that creates mprec
% quantities is also mprec

x=mprec(1); a=mprec(1); b=mprec(-6);
c=mprec(-72); d=mprec(-34.375);
for i = 1:33
    x = -(a*x^3 + b*x*x + d)/c;
end;
x
```

The cubic has the root $-1/2$. The output from the program is

$$-0.5000000000000000000000000000000000000000 \; .$$

4.2 Fixed-step Runge-Kutta code

An interesting demonstration of the ease of programming and a good test of the accuracy of the implementation is provided by classical fixed-step Runge-Kutta examples. As will be seen, with the aid of the `mpreal` class, the Matlab code for the classical Runge-Kutta algorithm appears to be identical to the code in double precision. Of course, once the variables are defined to be of class *mprec*, the operation take place at the higher precision.

Let a first-order ODE with a given initial value is written in the form,

$$\frac{dx}{dt} = f(t, x), \quad t_0 < t < T \tag{2}$$

$$x(t_0) = x_0 \tag{3}$$

where $x(t)$ is the unknown function.

An elementary code to calculate $x(T)$ is as follows

```
function x = rk4(fun, x0, t0, T, h)
% fun must be a function handle
n=floor((T-t0)/h);
t = t0;
x=x0;
for i = 1:n
 s1 = h*fun(t           , x);
 s2 = h*fun(t + 0.5*h, x + s1*0.5);
 s3 = h*fun(t + 0.5*h, x + s2*0.5);
 s4 = h*fun(t + h      , x + s3);
```

```
 x = x + (s1 + 2*(s2 + s3) + s4) / 6;
 t = t0 + i * h;
end
```

We can test our new class by solving the problem

$$\frac{dx}{dt} = \frac{tx - 2x^2}{t^2}, \quad 1 < t \leq 3 \tag{4}$$

$$x(1) = 4 \tag{5}$$

which has the analytical solution

$$x(t) = \frac{t}{0.25 + 2\ln(t)} \tag{6}$$

The Matlab code to test this is below. Note that a typical path has been shown, but this would be changed according to local storage.

```
path(path, 'My Documents\Matlab');
% t0, T are limits of integration
% x0 is an initial condition
% h is the step size.
t0 = mpreal(1.0);
T = mpreal(3.0);
x0 = mpreal(4.0);
n= mpreal(60000);
h = (T - t0) / n;
f = @(t,x) (t*x - 2*x*x)/(t*t);
x = rk4(f, x0, t0, T, h);

fe = @(t) t /(0.25 + 2*log(t));
exact = fe(T);
norm = abs(exact - x)
```

This gives the following output, which has been abbreviated to fit the page.

```
x=1.22587850244029152184455210127278 7
exact=1.22587850244029152153018542297
norm=3.14366678302186782313497*E-19.
```

5 Floating point function evaluation

When the precision of a calculation changes, it becomes important to be clear on the evaluation model being used. In regions where a problem is badly conditioned, this can be critical to understanding results.

The model used here is based on the IEEE standard and is also used by several commercial numerical products. Under the standard, any floating-point datum given to a function as an argument is regarded as exact. Another way of saying this is that the datum can be padded with an infinite number of zeros after the last digit supplied. This has the effect of meaning that transcendental constants such as π will be approximated by numbers slightly greater or slightly less than π depending upon the precision. For example, at 15 decimal digits, the approximation for π is slightly low (3.14159265358979), whereas at 20 decimal digits, the approximation is slightly high (3.14159265358979323 85). The IEEE double precision binary approximation is slightly low (equivalent in decimal digits to 3.14159265358979311 60).

We now consider three iterative schemes for the evaluation of Lambert W in regions where the problem is ill-conditioned. The three iterations are Halley's method, quoted above as equation (1), and two forms of Newton iteration:

$$w = w - \frac{we^w - z}{(1+w)e^w}\,, \tag{7}$$

$$w = w - \frac{w - ze^{-w}}{(1+w)}\,. \tag{8}$$

We know that W has an exact value at $z = -\pi/2$, namely $W_0(-\pi/2) = i\pi/2$. If this argument is perturbed slightly in the complex plane, say to $z = -\pi/2 + 10^{-6}i$, then calculations above show that the real part of the value is badly conditioned. We now revisit this calculation, first showing that the conditioning is independent of the iteration used. We computed the value of W using the three schemes above. The rate of convergence is higher using Halley than the other schemes. Using double precision and a starting estimate of $W \approx i$, Halley needs only 3 iterations to obtain the imaginary part carrect to all places; the real part, however, was correct only to 8 decimal digits and the remaining 8 digits oscillated with each iteration. The two Newton iterations took 6 iterations to reach the same accuracy, but the accuracy was essentially the same for all methods. Interestingly, scheme (8) managed one extra digit of accuracy in the real part.

The working precision was now raised to 30 digits, and the computations repeated. Now (1) converged to 30 digits in 4 iterations, while the Newton schemes took 7 iterations. The real part was accurate to 23 digits, with all schemes causing the last 7 digits to oscillate.

From these observations, one might think that computing the results to 30 digits and then rounding back to 16 would be the correct strategy, since all schemes obtained 23 digits correct. There is one proviso with this strategy, according to the floating-point model discussed above. If the user has asked for an evaluation to 16 digits, then the model above requires that the arguments should first be rounded to the 16 digits requested by the user, followed by the iterative calculation. The difference is as follows. Let z_{16} be $-\pi/2 + 10^{-6}i$ rounded to 16 digits, and z_{30} the same

quantity rounded to 30 digits. Then by computing to 30 digits in both cases, the 16 digit answers are

$$\begin{aligned} W(z_{16}) &= 2.884005766737188 \times 10^{-7} + \\ & \quad 1.5707958737766 87i \\ W(z_{30}) &= 2.884005769542418 \times 10^{-7} + \\ & \quad 1.5707958737766 87i \end{aligned}$$

Notice that the two evaluations differ precisely where the iterative schemes failed to converge. This again underlines the fact that the inaccuracies discussed here are results of the underlying mathematical definitions, and not simply computational errors.

6 Concluding Remarks

The AP precision function library, as collection of Matlab m-files, are presented. With the assistant of the implemented mpreal class, we presented arithmetic and algebraic operations in a natural notation. For maximum efficiency many of the Matlab functions in AP are implemented using C++ interfacing to Matlab.

We illustrated some classical algorithms, such as Newton's iterations, Lambert W function, and Runge-Kutta algorithm for ordinary differential equations. There are several extensions that can be consider for future work, one is to comprise some other fundamental algorithms and matrix mathematics, including eigensystems and matrix exponentials. The AP calculations using Matlab library is a relevant avenue as well.

Table 1: Basic arbitrary precision arithmetic and algebraic operations implemented in *mpreal* class

Function	Data Type
z = x + y	x, y, z AP complex
z = x - y	x, y, z AP complex
z = x * y	x, y, z AP complex
z = x / y	x, y, z AP complex
z = sin(x)	x, z AP real
z = cos(x)	x, z AP real
z = arcsin(x)	x, z AP real
z = arccos(x)	x, z AP real
z = arctan(x)	x, z AP real
z = exp(x)	x, z AP complex
z = ln(x)	x, z AP real
z = sqrt(x)	x, z AP real
z = x^n	x, z AP real, n integer
z = x^y	x, y, z AP real
$==, \sim=$	x, y, z AP real
$<, >, <=, >=$	AP real
z = double(x)	x string, z double
z = char(x)	x AP, string, z AP complex

References:

[1] http://crd.lbl.gov/∼dhbailey/mpdist.

[2] Bailey, D.H., 2005, High-Precision Floating-Point Arithmetic in Scientific Computation, Computing in Science and Engineering 7, 54-61.

[3] Bailey, D.H., Hida, Y., Li, X.S., Thompson, B., 2002, Arprec: An arbitrary precision computation package. http://crd.lbl.gov/˜dhbailey/dhbpapers/arprec.pdf.

[4] Boychuk, D., Braun, W.J., Kulperger, R.J., Krougly, Z.L, Stanford, D.A.,2009, A Stochastic Forest Fire Growth Models, Environmental and Ecological Statistics 16, 133-151.

[5] Brent, R.P., 1978. A Fortran multiple precision arithmetic package, ACM Transactions on Mathematical Software 4, 57-70.

[6] Borwein, J.M., Bailey, D.H., 2004. Mathematics by Experiment: Plausible Reasoning in the 21st Century, A K Peters Ltd, Natick, Massachusetts, 288 pp.

[7] Casale, G., Serazzi, G., 2006. Stabilization techniques for load-dependent queueing networks algorithms, in J.A.Barria Ed., Communication Networks and Computer Systems, Chapter 8, 127 - 141, Imperial College Press, London.

[8] Corless, R.M, Gonnet, G.H., Hare, D.E.G., Jeffrey, D.J., Knuth, D.E., 1996. On the Lambert W function, Advances in Computational Mathematics 5, 329-359.

[9] Dekker, T.J., 1971. A floating-point technique for extending the available precision, Numerische Mathematik 18, 224-242.

[10] Jeffrey, D.J., Hare, D.E.G., Corless, R.M., 1996. Unwinding the branches of the Lambert W function. The Mathematical Scientist 21, 1-7.

[11] Knuth, D.E., 1998. The Art of Computer Programming, volume 2: Seminumerical algorithms. Addison-Wesley, Reading MA, 3rd edn.

[12] Krougly Z.L., Creed, I.F., Stanford, D.A., 2009, A Stochastic Model for Generating Disturbance Patterns within Landscapes, Computers & Geosciences 35, 1451-1459.

[13] Krougly, Z.L., Stanford, D.A., 2005. Iterative algorithms for performance evaluation of closed network models, Performance Evaluation 61, 41-64.

[14] Shewchuk, J.R., 1997. Adaptive precision floating-point arithmetic and fast robust geometric predicates, Discrete & Computational Geometry 18, 305-363.