World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:6, No:1, 2012

Online computing system for octuple-precision computation with Fortran

Takemitsu Hasegawa and Yohsuke Hosoda

Abstract—Computations with higher than the IEEE 754 standard double-precision (about 16 significant digits) are required recently. Although there are available software routines in Fortran and C for high-precision computation, users are required to implement such routines in their own computers with detailed knowledges about them. We have constructed an user-friendly online system for octuple-precision computation. In our Web system users with no knowledges about high-precision computation can easily perform octuple-precision computations, by choosing mathematical functions with argument(s) inputted, by writing simple mathematical expression(s) or by uploading C program(s). In this paper we enhance the Web system above by adding the facility of uploading Fortran programs, which have been widely used in scientific computing. To this end we construct converter routines in two stages.

Keywords—Fortran, numerical computation, octuple-precision, Web.

I. Introduction

HIGHER than the IEEE-754 [9] 64-bit double-precision (about 16 significant digits) computations on computers are recently needed in scientific applications, say in high-energy physics [5] and numerical mathematics [7], see Bailey and Borwein [1]. Software packages for high-precision computation have been available [2], [4], [8], [10], [11], see [1] for detail. These packages require users to implement them on their own computers and to write individual programs with ample knowledges about the use of the packages. This means that it is not easy for non-experts on high-precision numerical computation to write programs by using the packages.

To facilitate the programming for high-precision computations we have constructed an user-friendly system on the Web for octuple-precision (about 72 significant digits) computation [6]. Our system makes use of the octuple-precision computation system ('octo C++') in C++ constructed by I. Ninomiya, see § II. Our system provides users with some facilities on the Web: one can 1) choose mathematical functions with arguments inputted, 2) write simple mathematical expression(s) in the form of the Web page and 3) upload the C program, see § III.

Fortran has a long history to be a widely used programming language in scientific computations on computers. In this paper we enhance our Web system above by adding the facility of uploading programs written in Fortran [12]. This facility enables users to perform the octuple-precision computation simply by writing a Fortran program in the IEEE 754 standard double-precision arithmetic. Our system doesn't require users to have any knowledges of the octuple-precision arithmetic. To this end we construct converter routines in two stages. In the first stage the Fortran program uploaded is converted into C program and the routine in the second stage converts the C program into octo C++ program, see § IV.

Dedicated to the memory of Professor Ichizo Ninomiya.
T. Hasegawa is with Department of Information Science, University of Fukui, Fukui, 910-8507, Japan e-mail:hasegawa@fuis.fuis.u-fukui.ac.jp.
Y. Hosoda is with University of Fukui.
Manuscript received November 11, 2011

II. Ninomiya's octo system in C++

Ichizo Ninomiya constructed a software package for octuple-precision (72 significant digits) computation in C++ ('octo C++', or 'octo system'). His packages perform computations very efficiently because fundamental mathematical functions such as $\sin x$ and e^x are approximated in terms of the minimax approximation. Two data types of floating-point numbers, namely, octuple-precision real ('octo') and octuple-precision complex ('ocmplx') are available. Four arithmetic operations ('$+, -, *, /$) of two octuple-precision numbers are possible. Fundamental mathematical functions are provided. A routine ('print') for outputting octuple-precision numbers is also available.

A. Octuple-precision real floating-point number

An octuple-precision real floating-point number('octo real') R is composed of eight 4-byte ($8 \times 4 \times 8 = 256$ bits) unsigned integers. The first bit is for the sign S followed by 15 bits for the exponent E and 240 bit for the mantissa M,

$$R = (-1)^S (1 + M) \times 2^{E-16383}.$$

By R one can express a floating-point number of more than 72 significant digits with the mantissa M of $240+1$ bits including one hidden bit. Table I compares the significant digits of the IEEE 754 single-precision (float) and double-precision (double) and present octuple-precision (octo) numbers.

TABLE I
Comparison of data types of float, double and octo

data type	sign (bit)	exponent (bit)	mantissa (bit)	significant digits
float	1	8	23	7
double	1	11	52	15
octo	**1**	**15**	**240**	**72**

1) Declaration and conversion of data types: One can declare variables and arrays in the octo type and convert the data types between integer, float, double and octo. By writing

```
octo a, b[5];
```

we can declare an octo real variable 'a' and an array 'b[5]'.

To convert 'a' above into integer, float and double types we write

int(a); float(a); double(a);

respectively. To convert 'i', 'f' and 'd' defined by

int i; float f; double d;

into the octo type we write

octo(i); octo(f); octo(d);

respectively.

2) Arithmetic operations: Arithmetic operations of four types, namely, addition, subtraction, multiplication and division of two octo real numbers are possible like those of double precision numbers. The mixed-mode operation of two numbers with different types of precisions are possible. The power of an octo variable is also available. To compute a^i we write

opow(a, i);

3) Mathematical constants: Important constants frequently used like π, $\log(2)$ and $\sqrt{2}$ and table functions like $\zeta(n)$ are available in octo type.

4) Mathematical functions: More than fifty octo real functions such as $\sin(x)$, $\log(x)$, $\exp(x)$, 10^x and $\Gamma(x)$ are available.

5) Output routine 'print': The routine '`print`' outputs computed results in about 72 significant digits. If we write

x = octo(pi) * octo(pi) + osqrt(2);
print(0, 0, 72, 1, x);

then we get the output result of $\pi^2 + \sqrt{2}$ as follows,

1.12838 17963 46245 36676 36179 72408 58492
13883 37128 26177 38699 59002 91142 10777e+01

Above 'print' routine feeds 0 line before printing octo-real 'a' in 0 blank followed by 72 significant digits with 5 consecutive digits together followed by 1 blank and finally followed by the exponent e+01.

B. Octuple-precision complex floating-point number

We can declare octuple-precision complex ('ocmplx') variable c and array w[7] by writing

ocmplx c, w[7];

The arithmetic operations, $+, -, \times, /$, of two 'ocmplx' numbers are possible. The mixed-mode operation of 'ocmplx' numbers, integers and double-precision numbers are possible. The power of 'ocmplx' variable is available, say, we compute z^i by writing

copow(z, i);

Octuple-precision complex functions such as $\sin(z)$, $|z| = \sqrt{x^2 + y^2}$ (where $z = x + iy$) and $\exp(z)$ are provided.

III. Online system for octuple-precision computation

In this section we review our online system for octuple-precision computation. Our system consists of client and server system, see Figure 1. On the server machine a CGI(Common Gateway Interface) program written in Perl receives a request from an user in the input form on the Web page. Then the CGI sends it to the computation program to ask for the computation requested. If the computation is successful, then the CGI sends the computed results to the client.

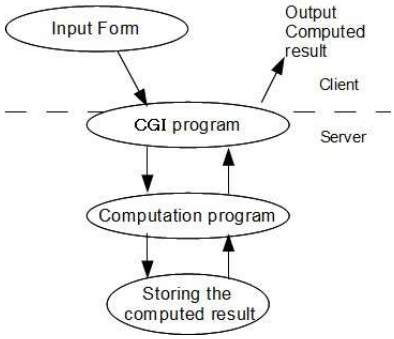

Fig. 1. Online octo system consisting of client and server system

The web pages on the server machine consists of an user-interface page, a manual page and a list of available functions.

A. User-interface

The user-interface page has input forms of three types.

1) Choosing functions: The first form allows one to choose a mathematical function in the pull-down menu, say, 'ocos' and to input an argument, say, 'pi/5'. Then one gets a computed result of $\cos(\pi/5)$ in 71 significant digits as follows,

8.09016 99437 49474 24102 29341 71828 19058
86015 45899 02881 43106 77243 11352 63023e-01

This result could be stored on the server and used again in the succeeding computation of other functions.

2) Writing mathematical expressions: The second form allows one to write simple mathematical expressions. If you write

osqrt(3.5) + oexp(2) / otan(pi/3)

then you get a computed result of $\sqrt{3.5} + e^{2.0}/\tan(\pi/3)$

6.13690 22211 61827 62558 89858 19692 67738
46106 00579 81936 89626 84375 28758 08091e+00

3) Uploading C programs: The third form is for some experts on the programming in C or C++. In this form one can upload a C(C++) program in the double-precision arithmetic. The system converts the uploaded program into an octo C++ program and implement it on the server to get computed results in 72 significant digits. If the computation is successful, then the results are sent back to the client.

B. Manual page and list of functions

A manual page in Japanese briefly illustrates how to input arguments of functions, possible data types and arithmetic operations and that nested mathematical expressions are possible. A list of available functions is provided. Figure 2 depicts a subset of the function list.

C. Computation process on the server

1) CGI program: The CGI program on the server receives data or a program file from the client and sends the results to the client. Assume that one selects a mathematical function in the pull-down menu on the web page and inputted value(s)

function	use
oabs	OCTUPLE ABSOLUTE VALUE FUNCTION
ocbrt	OCTUPLE CUBIC ROOT FUNCTION
oexpm1	oexpm1(x)=oexp(x)-1
oexp10	OCTUPLE BASE 10 EXPONENTIAL FUNCTION y = exp10(x)
ocos	OCTUPLE COSINE FUNCTION y = cos(x)
ocosq	OCTUPLE QUADRANT COSINE FUNCTION y = cosq(x) = cos((pi/2)*x)
ocot	OCTUPLE COTANGENT FUNCTION y = cot(x)
ocotq	QUADRUPLE QUADRANT COTANGENT FUNCTION y = cotq(x) = cot((pi/2)*x)
oacos	OCTUPLE ARCCOSINE FUNCTION y = acos(x)
olog	OCTUPLE LOGARITHMIC FUNCTION
olog2	OCTUPLE BASE 2 LOGARITHMIC FUNCTION y = log2(x)
osinh	OCTUPLE HYPERBOLIC SINE FUNCTION y = sinh(x)
otanh	OCTUPLE HYPERBOLIC TANGENT FUNCTION y = tanh(x)
oacosh	OCTUPLE INVERSE HYPERBOLIC COSINE FUNCTION y = acosh(x)
oerf	OCTUPLE ERROR FUNCTION y = erf(x) b
oceli1	complete elliptic integral of the first kind
olgama	OCTUPLE LOGARITHM OF GAMMA FUNCTION y=lgama(x)=log(gamma(x))
orgama	OCTUPLE RECIPROCAL GAMMA FUNCTION y=rgama(x)=1/gamma(x)
olfctr	OCTUPLE LOGARITHM OF FACTORIAL FUNCTION
offctr	OCTUPLE DOUBLE FACTORIAL FUNCTION n!!
oberno	OCTUPLE BERNOULLI NUMBERS b[n]=B(2n)
ogamco	OCTUPLE EXPANSION COEFFICIENTS OF 1/GAMMA(X+1)
oagmco	OCTUPLE ASYMPTOTIC EXPANSION COEFFICIENTS OF LOG(GAMMA(X))
obetno	OCTUPLE BETA NUMBERS

Fig. 2. A part of the list of available octo functions

of the argument in an input form on the web. Then the CGI starts by checking if some errors exist in the argument. If no error is detected, the CGI sends the inputted data to the computation program written in C++. On the other hand, when one writes simple mathematical expressions in a form on the web, the CGI program performs the similar procedure to send a computed result in 72 significant digits to the client.

2) Computation program: The compiled executable file 'a.out' of the computation program on the server has three arguments. The first argument receives the selected function name. The second argument receives the inputted value of the function argument, namely, the value of the function argument transformed by CGI program into a form suitable for the computation program. The third argument receives an user-decision result if computed results are stored in the server to be used in the succeeding computation or not. If the above procedure is successful, then 'a.out' is executed to transform the inputted strings into octuple-precision floating-point numbers or octo arithmetical expressions to perform the octo computation.

D. Converter routine for C or C++ program

Users can upload their C or C++ program in the double-precision arithmetic. To this end we have constructed a converter routine to translate the inputted program in C(C++) into an octuple-precision program.

1) Before translation: The converter routine starts by storing character strings of the inputted program file line by line into an array 'scr_line'. Scanning characters in 'scr_line' one by one with blanks and tabbing characters skipped the converter stores the detected types of the characters into a class of 'int' type. By using the detected types of characters the converter translates the inputted program into octo-program in C++. Three header lines required by octo-system are included

```
#include<iostream>
#include"octo.h"
#using namespace std;
```

before the translation starts, as follow.

2) Translation process: Double-precision floating-point numbers are converted into octo numbers, say,

$$a = 15.24 \Rightarrow a = \text{octo}(1524) * \text{opow}(\text{octo}(10), -2)$$

Mathematical functions are translated into the ones of octo type, say,

$$\text{sqrt}(5) \Rightarrow \text{osqrt}(\text{octo}(5))$$

The output routine 'printf' is translated by using 'print', say,

```
printf("a=%d, b=%f \n",a, b);
  ⇒  printf("a=%d", a);   printf(", b=");
     print(0, 5, 70, 1, d);   printf("<BR>");
```

3) Security check: It is important to protect the server from attacking outside, or malicious user-programs. The converter stops the translation and an alarm message is given on the web page if thirty five functions such as system functions and exec functions are detected.

4) Protection against endless loop: Successfully translated program file is compiled and executed to perform the octuple-precision computation. If the execution doesn't terminate in a prescribed time period, the computation is forced to stop and an error message is given on the web page to avoid the possible happening of an endless loop.

IV. Facility of uploading Fortran program

In this section we enhance our online system described in the previous section by adding the facility of uploading Fortran programs in the double-precision arithmetic. To this end we construct a converter program to translate Fortran programs uploaded by clients into the octo C++ program. Our converter consists of two steps. The first step makes use of the 'f2c' [3] to translate the Fortran program into C program. The second one translates the C program into the octo C++ program.

A. Complex variables and arithmetic operations

The arithmetic operations of complex variables are possible in Fortran and C++. But, ordinarily the C language doesn't accommodate to such operations. In this connection, the 'f2c' routine manages to translate Fortran programs including the complex arithmetic into the corresponding C program. To this end the routine makes use of reserved character strings and variable names as shown below.

1) Reserved names for complex variables: The 'f2c' reserves some character strings and variable names in the translated C program to accommodate the complex variables and arithmetic as well as integer and real ones used in the Fortran language. For example, 'integer', doublereal' and 'doublecomplex' are used for 'INTEGER', DOUBLE PRECISION' and 'DOUBLE COMPLEX' in Fortran.

2) Arithmetic operations of complex variables: The 'f2c' translates every complex variable(and constant) in Fortran by storing the real and imaginary parts of the variable separately in C program, for example,

d=e in Fortran ⇒ d_.r=e.r; d_.i=e.i;

Other Fortran statements such as

d=cmplx(b,c), d=d*e, d=d*2.3, d=(1.2, 3.4), b=real(e)

are translated into the corresponding statements in C.

TABLE II
COMVERSION OF COMPLEX DIVISIONS FROM FORTRAN TO C AND OCTO C++

Fortran	C by the f2c	octo C++
a = a / b;	c.div(&q_1, &a,&b); a.r = q_1.r, a.i = q_1.i;	q_1 = a / b; a = q_1;
c = c / d;	z.div(&z_1,&c, &d_); c.r = z_1.r, c.i = z_1.i;	z_1 = c / d_; c = z_1;

Table II shows how the divisions of two complex variables in Fortran are translated into those in octo C++ via in C. The statements in the third column show the divisions translated from C into octo C++. The second and third rows show the case of the division of two variables in complex type and in the fourth and fifth rows are that in double complex type. These two types of the division shown in the second column are not available in octo C++. We design our converter routine to cope with these divisions.

3) Conversion of complex functions: The 'f2c' routine converts 'CABS(N)' in Fortran to 'c_abs(&n)' in C. Further, we convert this to 'coabs(n)' in octo C++.

```
      integer a, g(5)
      real b
      real*8 c
      complex d, i
      complex*16 e
      character h
      data g / 2, 4, 6, 8, 10 /
      a = 5
      b = 6.7
①     d = (2.4, −3.5)
      h = 'm'
      i = (−3.3, 4.4)
②     d = d / i
③     i = abs(i)
④     write(*,*) b
      write(*,*) g
      end
```

Fig. 3. An example of Fortran program to be converted

B. Output routine

The 'f2c' converts output statements in Fortran to C as follows,

```
write(*,*) a   ⇒
      s_wsle(&io_7));
      do_lio(&c_3,&c_1,(char *)&a,(ftnlen)sizeof(integer));
      e_wsle();
```

We convert these statements to octo C++ as follows

⇒ printf("%d",a); printf("\n");

```
/* 2.f -- translated by f2c (version 20061008).
   You must link the resulting object file with libf2c:
        on Microsoft Windows system, link with libf2c.lib;
        on Linux or Unix systems, link with .../path/to/libf2c.a -lm
        or, if you install libf2c.a in a standard place, with -lf2c -lm
        -- in that order, at the end of the command line, as in
                cc *.o -lf2c -lm
        Source for libf2c is in /netlib/f2c/libf2c.zip, e.g.,
                http://www.netlib.org/f2c/libf2c.zip
*/                                                                    ⑦
#include "f2c.h"
/* Table of constant values */
static integer c__4 = 4;
static integer c__1 = 1;
static integer c__3 = 3;

   static integer c__5 = 5;
   /* Main program */ int MAIN__(void)
   {
       /* Initialized data */
       static integer g[5] = { 2,4,6,8,10 };
       /* System generated locals */
       real r__1;
       complex q__1;
⑤      /* Builtin functions */
       void c_div(complex *, complex *, complex *);
       double c_abs(complex *);
       integer s_wsle(cilist *), do_lio(integer *, integer *, char *, ftnlen),
               e_wsle(void);
       /* Local variables */
       static integer a;
       static real b;
       static complex d__;
       static char h__[1];
       static complex i__;
       /* Fortran I/O blocks */
⑥      static cilist io___7 = { 0, 6, 0, 0, 0 };
       static cilist io___8 = { 0, 6, 0, 0, 0 };

       a = 5;
       b = 6.7f;
①      d__.r = 2.4f, d__.i = -3.6f;
       *(unsigned char *)h__ = 'm';
       i__.r = -3.3f, i__.i = 4.4f;
②      /* 複素数同士の除算---------------! */
       c_div(&q__1, &d__, &i__);
       d__.r = q__1.r, d__.i = q__1.i;
③      /* 算術関数-----------------------! */
       r__1 = c_abs(&i__);
       i__.r = r__1, i__.i = 0.f;
④      /* 出力文-------------------------! */
       s_wsle(&io___7);
       do_lio(&c__4, &c__1, (char *)&b, (ftnlen)sizeof(real));
       e_wsle();
   /* -------------------------------! */
       s_wsle(&io___8);
       do_lio(&c__3, &c__5, (char *)&g[0], (ftnlen)sizeof(integer));
       e_wsle();
   /* -------------------------------! */
       return 0;
   } /* MAIN__ */
```

Fig. 4. A program in C translated by the 'f2c' routine from the Fortran program in Fig 3

```
#include <iostream>
#include "okansu.h"                                    ⑧
using namespace std;
static int i__c;
/* 2.f -- translated by f2c (version 20061008).
   You must link the resulting object file with libf2c:
        on Microsoft Windows system, link with libf2c.lib;
        on Linux or Unix systems, link with .../path/to/libf2c.a -lm
        or, if you install libf2c.a in a standard place, with -lf2c -lm
        -- in that order, at the end of the command line, as in
                cc *.o -lf2c -lm                                        ⑦
        Source for libf2c is in /netlib/f2c/libf2c.zip, e.g.,
                http://www.netlib.org/f2c/libf2c.zip
*/
#include "f2c.h"
/* Table of constant values */
static int c__4 = 4;
static int c__1 = 1;
static int c__3 = 3;
static int c__5 = 5;
/* Main program */ int main(void)
{
    /* Initialized data */
    static int g[5] = { 2,4,6,8,10 };
    /* System generated locals */
    octo r__1;
    ocmplx q__1;
    /* Local variables */
    static int a;
    static octo b;
    static ocmplx d__;
    static char h__[1];
    static ocmplx i__;
    /* Fortran I/O blocks */

    a = 5;
    b = octo(67)*opow(octo(10),-1) ;
①   d__ = ocmplx(octo(24)*opow(octo(10),-1) ,-octo(36)*opow(octo(10),-1) );
    *(unsigned char *)h__ = 'm';
    i__ = ocmplx(-octo(33)*opow(octo(10),-1) ,octo(44)*opow(octo(10),-1) );
②/* 複素数同士の除算---------------! */
    q__1 = d__ / i__ ;
    d__ = q__1;
③/* 算術関数-----------------------! */
    r__1 = coabs(i__);
    i__ = ocmplx(r__1,octo(0)*opow(octo(10),-1) );
④/* 出力文-------------------------! */

    print(0,5,70,1,b);printf(" ");
    printf("¥n");
/* -------------------------------! */

    for(i__c=0;i__c<c__5;i__c++){
      printf("%d",g[i__c]);
    }
    printf("¥n");
/* -------------------------------! */
    return 0;
} /* main */
```

図 1:8倍精度 C++プログラム

Fig. 5. An octo C++ program translated from the C program in Fig 4

Our converter uses the 'for' loop to convert the output routine for arrays, see ④ and the following sentences in Fig 5 below.

C. An example of translation

Fig 4 shows a program in C translated by the 'f2c' routine from a Fortran program shown in Fig 3. On the other hand, Fig 5 shows a program in octo C++ translated by our converter routine from the program shown in Fig 4. The sentence on a complex constant marked by ① in Fig 3 is translated into ① in Fig 4 and further into ① in Fig 5. Similarly, the sentences marked by ② ∼ ④ in Fig 3 are translated.

V. Conclusion

We constructed an use-friendly online system for octuple-precision computation. Our system makes use of Ninomiya's octuple-precision computation system written in C++(octo C++). Our Web system requires users no knowledges about high-precision computation. By our system users can perform octuple-precision computation simply by choosing mathematical functions with values of arguments inputted, by writing simple mathematical expressions in the Web form and by uploading their own C or C++ program in double-precision arithmetic. Furthermore, we added a facility of uploading Fortran programs. To this end we constructed a converter routines from Fortran to octo C++.

In summary, our system enables remote users to perform octuple-precision computation without any knowledges about high-precision computations,

Acknowledgements

We thank Ichizo Ninomiya (Emeritus Professor of Nagoya University) for providing us with his software packages for octuple-precision computation. This research is supported in part by Grants-in-Aid for Scientific Research (C)21540123 of JSPS (Japan Society for the Promotion of Science).

References

[1] D. H. Bailey and J. M. Borwein, *High-precision computation and mathematical physics*, http://crd.lbl.gov/~dhbailey/dhbpapers/dhb-jmb-acat08.pdf, 2009
[2] R. Crandall and J. Papadopoulos, *Octuple-precision floating point on Apple G4*, http://image.apple.com/acg/pdf/oct3a.pdf, 2002.
[3] S. I. Feldman, D. M. Gay, M. W. Maimore and N. L. Schryer, A Fortran-to-C Converter, *Computing Science Technical Report* No. 149, 1990, http://www.netlib.org/f2c
[4] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier and P. Zimmermann: MPFR: A multiple-precision binary floating-point library with correct rounding, *ACM Trans. Math. Soft.*, 33, 2007, 13:1–13:15.
[5] J. Fujimoto, N. Hamaguchi, et al., Numerical precision control and GRACE, *Nucl. Instr. Meth. Phys. Res. A*, 559, 2006, 269–272.
[6] T. Hasegawa, Y. Hosoda and K. Hirai, An online system for octuple-precision computation, *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks(PDCN2011)*, Innsbruck, 2011, 70–74.
[7] Y. Hatano, I. Ninomiya, H. Sugiura and T. Hasegawa, Numerical evaluation of Goursat's infinite integral, *Numerical Algorithms*, 52(2), 2009, 213-224.
[8] A. H. Karp and P. Markstein, High-precision division and square root, *ACM Trans. Math. Soft.*, 23, 1997, 561-589.
[9] M. L. Overton, *Numerical Computing with IEEE Floating Point Arithmetic*, Philadelphia: SIAM, 2001.
[10] D. M. Smith, A multiple-precision division algorithm, *Math. Comp.*, 65, 1996, 157–163.
[11] D. M. Smith, Algorithm 786 Multiple-precision complex arithmetic and functions, *ACM Trans. Math. Soft.*, 24, 1998, 358-367.
[12] http://netnumpac.fuis.fukui-u.ac.jp/~tanabe/cgi-bin/test/index.html

Takemitsu Hasegawa Takemitsu Hasegawa received the Doctor of Engineering degree in applied physics from Nagoya University, Japan in 1975. He is an emeritus professor at University of Fukui, Japan. He was a professor of Department of Information Science at University of Fukui from 1995 to 2010 after working as an associate professor at University of Fukui and as a research associate at Nagoya University. His research interests include numerical analysis, high performance computing, online system for mathematical software library and Web computation.

Yohsuke Hosoda Yohsuke Hosoda received the Doctor of Engineering degree in computer science from Nagoya University, Japan in 1994. He is a professor of Department of Information Science at University of Fukui. His research interests include numerical analysis such as ill-posed system of linear equations, inverse problem and numerical linear algebra.