Proc. of ASME Design Automation Conference, Albuquerque, NM, September 19-22, 1993, vol. 2, pp. 73-80.

COMPUTATIONS OF DUAL NUMBERS IN THE EXTENDED FINITE DUAL PLANE

Harry H. Cheng
Department of Mechanical, Aeronautical
and Materials Engineering
University of California
Davis, California

Abstract

The numerical computational aspects of dual numbers in the C^H programming language are presented in this paper. Dual is a built-in data type in C^H. Dual numbers and dual metanumbers are described in the extended dual plane and extended finite dual plane. The arithmetic and relational operations, and built-in mathematical functions are defined for both dual numbers and dual metanumbers of DualZero, DualInf, and DualNaN. Due to polymorphism, the syntax of dual arithmetic and relational operations, and built-in dual functions are the same as those for real and complex numbers in the C^H programming language. The linguistic features and handling of user's dual functions and dual arrays, as well as applications of dual numbers in robotics and spatial mechanisms in the C^H programming language can be found in (Cheng, 1993c).

1. Introduction

Clifford (1873) introduced dual numbers in the form of $x + \varepsilon y$ with $\varepsilon^2 = 0$ to form bi-quaternions (called dual quaternions nowadays) for studying the noneuclidean geometry. Study (1903) defined dual numbers as dual angles to specify the relation between two lines in the Euclidean space. McAulay (1898) used dual quaternions to describe finite displacement of rigid and deformable bodies. Dimentberg (1950) and Denavit (1958) pioneered kinematic analysis of spatial mechanisms via dual numbers. Using dual numbers, Yang and Freudenstein (1963, 1964) studied kinematic analysis of spatial mechanisms; and Ravani and Roth (1984) studied rigid body displacements via kinematic mapping. Recently, dual numbers have been applied to study the kinematics, dynamics, and calibration of open-chain robot manipulators (Pennock and Yang, 1985; 1983; Dooley and McCarthy, 1991; Ravani and Ge, 1991). Dual numbers are useful for analytical treatment in kinematics and dynamics of spatial mechanisms. Succinct formulas and equations can be derived by using dual numbers. However, the concise dual formulas cannot be conveniently implemented in a computer program using commonly used computer programming languages. The conciseness of a symbolic formula will diminish when numerical computations are needed. Therefore, the application of dual numbers in the study of problems in science and engineering has not been fully appreciated yet.

In this paper, we will demonstrate that programming with dual numbers can be as simple as programming with real or complex numbers, depending on the implementation of a computer language. FORTRAN was designed in the 1950s for FORmula TRANslation, but not for formulas with dual numbers. This language design reflects the state-of-the-art of computer science and technology in the 1950s and the perception of scientific computation at that time although many advances in the field have been incorporated into its standard in 1977 as FORTRAN77 (ANSI 1978). C, a modern computer language, has gained its wide acceptance as it was standardized recently (ANSI, 1989). C is most popular for system programming because it was originally designed for Unix system programming (Kernighan and Ritchie, 1978). The numerically-oriented scientific programming was not the original design goal; needless to say, the handling of dual numbers. Recently, we have designed and implemented the computer language C^H which is a general-purpose block-structured interpretive programming language. It retains most features of ANSI C, but with many High-level extensions. C^H combines features of many computer languages and software packages, and it is designed to be especially suitable for reasearch and applications in the areas of scientific and system programming as well as mechanical systems engineering. In most of its applications, the size of a program written in C^H is significantly smaller than that written in Fortran or C. The programming of C^H with real and complex numbers are presented in (Cheng, 1993a; 1993b). Like float, double, and complex, dual is implemented as a built-in data type in C^H. C^H is the first computer programming language, I believe, that provides dual as a basic data type. In the C^H language, formulas formulated in dual numbers can be translated into programs as easy as those expressed in real or complex numbers. The numerical computational aspects of dual numbers in the C^H programming language will be described in this paper.

2. Dual Numbers

2.1. Dual Numbers and Dual Constructor

Dual numbers $d \in \mathcal{D} = \{(x, y)|x, y \in \mathcal{R}\}$ can be defined as ordered pairs

$$d = (x, y) \tag{1}$$

with specific addition and multiplication rules (Clifford, 1873; Guggenheimer, 1963; Bottema and Roth, 1979). The real numbers x and y are called the *real* and *dual parts* of d, respectively. If both x and y of a dual number are nonzero, it is called a *proper dual* number. If we identify the pair of $(x, 0)$ as real numbers. The real number $\mathcal{R}$ is a subset of $\mathcal{D}$, i.e., $\mathcal{R} = \{(x, y)|x \in \mathcal{R}, y = 0\}$ and $\mathcal{R} \subset \mathcal{D}$. If a real number is considered either as x or $(x, 0)$ and let ε denote the *pure dual* number $(0, 1)$ with the property of $\varepsilon^2 = 0$, dual numbers can be written as

$$d = x + \varepsilon y \tag{2}$$

Resembling the notation (1), a dual number can be created in $\mathrm{C^H}$ by the *dual constructor* **dual**(x, y) with $x, y \in \mathcal{R}$. For example, a dual number with its real part of 1.2 and dual part of 3.4 can be constructed by dual(1.2, 3.4). Internally, a dual number consists of two floats at the current implementation. Hence, arguments of the dual constructor dual(x,y) will always be cast to floats internally if they are not floats. The internal data representation of a dual number is similar to that of a complex number. For the convenience of presentation and programming, the second element of a dual number is, therefore, also referred to as *imaginary part* in $\mathrm{C^H}$. The names of *dual part* and *imaginary part* are used interchangeably in this paper.

It should be emphasized that dual numbers are extension of real numbers. If the dual parts of dual numbers are identically zeros, results of dual operations and dual functions should be dual numbers with identical zero dual parts. This design goal has been achieved in $\mathrm{C^H}$. For example, **sqrt**(dual(9.0, 0.0)) equals dual(3.0, 0.0) and **pow**(dual(2.0,0.0), dual(3.0,0.0) becomes dual(8.0, 0.0), all with identically zero dual parts. These dual functions will be discussed in details in section .

2.2. Dual Metanumbers and Geometric Interpretation

A dual number $d = x + \varepsilon y$ can be associated with a point in the plane whose Cartesian coordinates are x and y. Each dual number corresponds to just one point in the plane, and vice versa. The number d can also be thought of as the vector from the origin to the point (x, y). When used for the purpose of displaying numbers $d = x + \varepsilon y$ geometrically, we call the XY plane the *dual plane*, or *d plane*, the X-axis the *real axis*, and the Y-axis the *dual axis*.

We define the *dual conjugate* of the dual number $d = x + \varepsilon y$ as the dual number $x - \varepsilon y$, denoted as $\overline{d}$. The dual number $\overline{d}$ represented by the point $(x, -y)$ in the dual plane is the reflection in the real axis of the point (x, y) representing d.

As described in (Cheng, 1993a), $\mathrm{C^H}$ provides real metanumbers of 0.0, −0.0, Inf, −Inf, and NaN for scientific computing. Because of metanumbers of Inf and −Inf, $\mathrm{C^H}$ distinguishes 0.0 from −0.0 for real numbers in arithmetic operations and elementary mathematical functions. For proper handling of complex numbers, Cheng (1993b) introduced complex metanumbers of ComplexZero, ComplexInf, and ComplexNaN along with the extended finite complex plane. In this paper, we introduce the *extended dual plane* shown in Figure 1. The extended dual plane is the dual plane plus the point at infinity denoted as DualInf. In Figure 1, there is a one-to-one correspondence between the points on the unit sphere Γ and the points in the extended dual plane $\mathcal{D}$. The point p on the surface of the sphere is determined by the intersection of the line through the point d in th extended dual plane and the north pole N of the sphere. There is only *one* dual infinity in the extended dual plane, the north pole N corresponds to the point at infinity. Due to the finite representation of floating-point numbers, in this paper, we also introduce the *extended finite dual plane* shown in Figure 2. Any dual values inside the ranges of $|x|$ < FLT_MAX and $|y|$ < FLT_MAX in the extended finite dual plane are representable in finite floating-point numbers. FLT_MAX, a predefined system constant, is the maximum representable finite floating-point number in the float data type. Outside this rectangular area of the extended finite dual plane, a dual number is treated as a Dual-Infinity represented as DualInf or dual(Inf,Inf) in $\mathrm{C^H}$. The one-to-one correspondence between points on the unit sphere Γ and extended dual plane is not valid for the unit sphere Λ and extended finite dual plane. All points on the surface of the upper part Λ_1 of the unit sphere correspond

Figure 1: The extended dual plane and unit sphere Γ.

Figure 2: The extended finite dual plane and unit sphere Λ.

to the dual infinity. Points on the lower part Λ_2 of the sphere and points in the extended finite dual plane are in one-to-one correspondence. The boundary between surfaces Λ_1 and Λ_2 corresponds to the threshold of overflow. For example, points p_1 and p_2 on the unit sphere Λ correspond to points d_1 = dual(FLT_MAX, 0.0) and d_2 = dual(FLT_MAX, FLT_MAX) in the extended finite dual plane shown in Figure 2, respectively. The origin of the extended finite dual plane is dual(0.0, 0.0) or DualZero which stands for DualZero. It should be pointed out that, by defining the dual unit sphere as $\mathbf{d} \cdot \mathbf{d} = 1$ where $\mathbf{d} = \mathbf{a} + \varepsilon \mathbf{a}_0$ is a dual unit vector with $|\mathbf{a}| = 1$ and $\mathbf{a} \perp \mathbf{a}_0$, Study (1903) established the one-to-one correspondence between the oriented lines in space and the points of the dual unit sphere (Guggenheimer, 1963). However, that one-to-one correspondence is different from the one-to-one correspondence between points on the unit sphere and the extended dual plane presented in this paper.

Like a real number which can be considered as an angle, in differential geometry and motion analysis of spatial mechanisms (Study 1903; Guggenheimer, 1963; Yang and Freudenstein, 1964), a dual number is also commonly referred to as a *dual angle*

$$\hat{\theta} = \theta + \varepsilon s \tag{3}$$

between two lines L_1 and L_2 in space. The real part θ of the dual angle is the projected angle between lines L_1 and L_2, and the dual part s is the length along the common normal of two lines. In general, the dual angle between two non-parallel or non-intersecting lines in space is a proper dual number. The dual angle subtended by two intersecting lines is a real number. The dual angle between two parallel lines is a pure dual number. A dual angle can also be represented by

$$d = \hat{\theta} = \theta(1 + \varepsilon p) \tag{4}$$

where the ratio $p = s/\theta$ is referred to as the *pitch* of the dual angle $\hat{\theta}$. If the pitch p is zero, it is a pure rotation; if p is infinity, it is a pure translation. The values of $\pm\infty$ are represented by real metanumbers of $\pm$Inf in C^H (Cheng, 1993a).

In C^H, an undefined or mathematically indeterminate dual number is denoted as DualNaN or dual(NaN, NaN) which stands for Dual-Not-a-Number. To ensure the proper flow of the program, all operations and functions are defined over the domain of the extended finite dual plane. Furthermore, To guarantee the delivery of correct numerical results, a built-in dual operation or dual function in C^H will deliver DualNaN if it is mathematically indeterminate, e.g., **sqrt**(dual(-3.0, 0.0)) is DualNaN. We call special dual numbers of DualZero, DualInf, and DualNaN as *dual metanumbers*. Because of the mathematical infinities of $\pm\infty$, it becomes necessary to distinguish the positive zero of 0.0 from the negative zero of -0.0 for real numbers (Cheng, 1993a). Unlike the real line along which real numbers can approach the origin only in the positive or negative numbers, the origin of the dual plane can be reached in any directions. Therefore, dual operations and dual functions in C^H do not distinguish 0.0 from -0.0 for real and dual parts of a dual number. Due to these differences, some operations and functions need to be handled differently for real and dual numbers, especially for real metanumbers and dual metanumbers. For example, the addition of two real numbers of positive infinities is a value of infinity in C^H(Cheng, 1993a). But, the addition of two dual infinities is indeterminate although the value of DualInf is represented internally as two positive infinities of Infs.

Table 1: Dual operations.

Definition	C^H Syntax	C^H Semantics
negation	−d	$-x - \varepsilon y$
addition	d1 + d2	$(x_1 + x_2) + \varepsilon(y_1 + y_2)$
subtraction	d1 − d2	$(x_1 - x_2) + \varepsilon(y_1 - y_2)$
multiplication	d1 * d2	$x_1 * x_2 + \varepsilon(y_1 * x_2 + x_1 * y_2)$
division	d1 / d2	$\frac{x_1}{x_2} + \varepsilon \frac{y_1 * x_2 - x_1 * y_2}{x_2^2}$
equal	d1 == d2	$x_1 == x_2$ and $y_1 == y_2$
not equal	d1 != d2	$x_1 != x_2$ or $y_1 != y_2$

Table 2: Negation results.

Negation −				
operand	DualZero	d	DualInf	DualNaN
result	DualZero	−d	DualInf	DualNaN

2.3. Dual Functions

The *dual function* $f(d) = f(x + \varepsilon y)$ with a dual variable can be obtained by expanding it formally in a Taylor series. Because $\varepsilon^n = 0$ for $n > 1$, then,

$$f(d) = f(x + \varepsilon y) = f(x) + \varepsilon y f'(x) \tag{5}$$

where $f'(x)$ in the dual part is the derivative of function $f(x)$. Similarly, a dual function with two dual variables can be obtained as

$$\begin{aligned} (d_1, d_2) &= f(x_1 + \varepsilon y_1, x_2 + \varepsilon y_2) \\ &= f(x_1, x_2) + \varepsilon(y_1 f'_{d_1}(x_1, x_2) + y_2 f'_{d_2}(x_1, x_2)) \end{aligned} \tag{6}$$

where $f'_{d_1}(x_1, x_2)$ is the partial derivative with respect to the first variable d_1 and $f'_{d_2}(x_1, x_2)$ with respect to the second. Using these definitions, all identities for ordinary elementary functions hold true for dual functions with dual variables. For example, $\sin^2 d + \cos^2 d = 1$; $\cosh^2 d - \sinh^2 d = 1$, $d_1^{d_2} = \exp(d_2 \log(d_1))$. The built-in dual functions in C^H will be described in details in section .

3. Dual Operations

The arithmetic and relational operations for dual numbers are treated in the same manner as those for real and complex numbers in C^H. This section will discuss how these operations are defined and handled by C^H.

3.1. Dual Operations with Dual Numbers

The negation, arithmetic, and comparison operations for dual numbers are defined in Table 1, where the dual numbers d, d_1, and d_2 are defined as $x + \varepsilon y, x_1 + \varepsilon y_1$, and $x_2 + \varepsilon y_2$, respectively.

The negation of a dual number will change the sign of both real and dual parts of the dual number. The addition of two dual numbers will add real and dual components of two dual numbers, separately. The subtraction of two dual numbers will subtract real and dual parts of the second dual number from real and dual parts of the first dual number, respectively. Treating the pure dual ε as a dual number

Table 3: Addition and subtraction results.

Addition and Subtraction ±				
left operand	right operand			
	DualZero	d2	DualInf	DualNaN
DualZero	DualZero	±d2	DualInf	DualNaN
d1	d1	d1 ± d2	DualInf	DualNaN
DualInf	DualInf	DualInf	DualNaN	DualNaN
DualNaN	DualNaN	DualNaN	DualNaN	DualNaN

Table 4: Multiplication results.

Multiplication *				
left operand	right operand			
	DualZero	d2	DualInf	DualNaN
DualZero	DualZero	DualZero	DualNaN	DualNaN
d1	DualZero	d1*d2	DualInf	DualNaN
DualInf	DualNaN	DualInf	DualInf	DualNaN
DualNaN	DualNaN	DualNaN	DualNaN	DualNaN

Table 6: Equal comparison results.

Equal comparison ==				
left operand	right operand			
	DualZero	d2	DualInf	DualNaN
DualZero	1	0	0	0
d1	0	d1 == d2	0	0
DualInf	0	0	1	0
DualNaN	0	0	0	1

Table 7: Not equal comparison results.

Not equal comparison !=				
left operand	right operand			
	DualZero	d2	DualInf	DualNaN
DualZero	0	1	1	1
d1	1	d1 != d2	1	1
DualInf	1	1	0	1
DualNaN	1	1	1	0

of dual(0, 1), the multiplication and division for two dual numbers are defined in Table 1. Dual numbers form an algebra, not a field. The pure dual numbers are zero divisors such that dual(0.0,y1)$*$dual(0.0,y2) = dual(0.0,0.0). For binary operations with a real or complex operand and a dual operand, the real or complex operand will be promoted to a dual before the operation. One cannot compare whether one dual number is larger or smaller than other. But two dual numbers can be tested whether they are equal or not. Two dual numbers are equal to each other iff both real and dual parts of two dual numbers are equal to each other, separately. If real parts or dual parts of two dual numbers are not equal to each other, two dual numbers are said not equal.

3.2. Dual Operations with Dual Metanumbers

In the above definitions of dual operations, we assume that all operands are dual numbers. The real and dual parts of a dual number are then treated as two regular floating-point floats. If the values of operands involve dual metanumbers, the definitions defined in Table 1 may not be valid. For example, DualInf is represented internally as dual(Inf, Inf). According to the dual addition definition defined in Table 1 and addition rule for real numbers discussed in (Cheng, 1993a), the result of addition of two DualInfs would be dual(Inf, Inf). But, addition of two dual infinities is mathematically indeterminate, which is the value of DualNaN in $\mathrm{C^H}$. Division of a dual number by a pure dual can be either DualNaN or DualInf, depending on the value of the numerator. Results for arithmetic and relational operations with both dual numbers and dual metanumbers are defined in Tables 2 to 7.

From a programmer's point of view, values of dual($\pm 0.0, \pm 0.0$) are treated the same as DualZero or dual(0.0, 0.0) when they are used as operands or arguments in $\mathrm{C^H}$. In the following discussions, the positive zero of 0.0 and the negative zero of -0.0 for real and dual components of a dual number are considered the same. Therefore, although the negation of dual(0.0, 0.0) returns dual($-0.0, -0.0$), the result listed in Table 2 is DualZero. Negation of a dual infinity is still a dual infinity. Of cause, negation of a Dual-Not-a-Number is DualNaN.

For binary operations in Tables 3 to 5, if any one of operands is DualNaN, the result is DualNaN. If one of two operands is DualInf and other is a finite dual number, the result of addition and subtraction is DualInf. Unlike real numbers, addition and subtraction of two DualInfs are DualNaNs. Multiplication of DualInf with dual(0.0, 0.0) is DualNaN; multiplication of DualInf with a finite nonzero number is DualInf; and multiplication of two DualInfs becomes DualInf. Like real numbers, divisions of dual(0.0, 0.0) by dual(0.0, 0.0) and DualInf by DualInf are DualNaNs. A finite number or infinity divided by dual(0.0, 0.0) becomes DualInf. The division of DualInf by a finite number gives DualInf. Unlike complex numbers, the division a proper dual number by a pure dual dual(0.0, y) is formally defined as DualInf. Theoretically, two dual infinities cannot be compared with each other, they may or may not be equal to each other. In $\mathrm{C^H}$, however, two DualInfs are considered the same from the programming point of view as shown in Table 6. Likewise, the comparison of two DualNaNs will get a logic TRUE. This design consideration is also reflected in the `not equal` relational operation shown in Table 7.

4. Dual Functions

The external functions of $\mathrm{C^H}$ can be created in the same manner as C functions are created. Unlike C and Fortran, however, the commonly used mathematical functions are built internally in the language itself in $\mathrm{C^H}$. The mathematical functions in $\mathrm{C^H}$ can handle different data types of the arguments gracefully. The output data type and algorithm of a function depend on the data types of the input arguments, which is called *polymorphism*. Like arithmetic operators, the built-in commonly used mathematical functions in $\mathrm{C^H}$ are polymorphic. For example, if the order of the data type of the input argument x is less than float, function **sin**(x) computes the sine of x in float. If x is double, the

Table 5: Division results.

Division ÷					
left operand	right operand				
	DualZero	dual(0.0, y2)	d2	DualInf	DualNaN
DualZero	DualNaN	DualNaN	DualZero	DualZero	DualNaN
dual(x1, 0.0)	DualNaN	DualNaN	dual($\frac{x_1}{x_2}, -\frac{x_1 y_2}{x_2^2}$)	DualZero	DualNaN
dual(0.0, y1)	DualNaN	DualNaN	dual(0.0, $\frac{y_1}{x_2}$)	DualZero	DualNaN
d1	DualInf	DualInf	$\frac{d1}{d2}$	DualZero	DualNaN
DualInf	DualInf	DualInf	DualInf	DualNaN	DualNaN
DualNaN	DualNaN	DualNaN	DualNaN	DualNaN	DualNaN

Table 8: Syntax and semantics of built-in dual functions.

C^H Syntax	C^H Semantics
sizeof(d)	8
abs(d)	sqrt($x^2 + y^2$)
real(d)	x
imaginary(d)	y
dual(x, y)	$x + \varepsilon y$
conjugate(d)	$x - \varepsilon y$
polar(d)	sqrt($x^2 + y^2$) $+ i\,\Theta$
sqrt(d)	sqrt(x) $+ \varepsilon \frac{y}{2\mathrm{sqrt}(x)}$
exp(d)	$\exp(x)(1 + \varepsilon y)$
log(d)	$\log(x) + \varepsilon \frac{y}{x}$
log10(d)	$\frac{\log(d)}{\log(10)}$
pow(d_1, d_2)	${d_1}^{d_2} = x_1^{x_2} + \varepsilon(y_1 x_2 x_1^{x_2 - 1} + y_2 x_1^{x_2} \log(x_1))$
sin(d)	$\sin(x) + \varepsilon y \cos(x)$
cos(d)	$\cos(x) - \varepsilon y \sin(x)$
tan(d)	$\tan(x) + \varepsilon \frac{y}{\cos(x)\cos(x)}$
asin(d)	$\mathrm{asin}(x) + \varepsilon \frac{y}{\mathrm{sqrt}(1 - x^2)}$
acos(d)	$\mathrm{acos}(x) - \varepsilon \frac{y}{\mathrm{sqrt}(1 - x^2)}$
atan(d)	$\mathrm{atan}(x) + \varepsilon \frac{y}{1 + x^2}$
sinh(d)	$\sinh(x) + \varepsilon y \cosh(x)$
cosh(d)	$\cosh(x) + \varepsilon y \sinh(x)$
tanh(d)	$\tanh(x) + \varepsilon \frac{y}{\cosh(x)\cosh(x)}$
asinh(d)	$\mathrm{asinh}(x) + \varepsilon \frac{y}{\mathrm{sqrt}(x^2 + 1)}$
acosh(d)	$\mathrm{acosh}(x) - \varepsilon \frac{y}{\mathrm{sqrt}(x^2 - 1)}$
atanh(d)	$\mathrm{atanh}(x) + \varepsilon \frac{y}{1 - x^2}$
ceil(d)	$\mathrm{ceil}(x) + \varepsilon\ \mathrm{ceil}(y)$
floor(d)	$\mathrm{floor}(x) + \varepsilon\ \mathrm{floor}(y)$
ldexp($d1, d2$)	$\mathrm{ldexp}(x_1, x_2) + \varepsilon\ \mathrm{ldexp}(y_1, y_2)$
fmod($d1, d2$)	$d;\ \ \frac{d_1}{d_2} = k + \frac{d}{d_2},\ k \geq 0$
modf($d1, \&d2$)	$\mathrm{modf}(x_1, \&x_2) + \varepsilon\ \mathrm{modf}(y_1, \&y_2)$
frexp($d1, \&d2$)	$\mathrm{frexp}(x_1, \&x_2) + \varepsilon\ \mathrm{frexp}(y_1, \&y_2)$

returned data is in double. The same function **sin**(x) will give complex or dual result if x is complex or dual, respectively. Hence, one will not need to remember many arcane names as in Fortran or C. For the portability, all mathematical functions included in the ANSI C header `math.h` have been implemented polymorphically in C^H. The names of built-in mathematical functions of C^H presented in this paper are based upon the ANSI C header `math.h`. However, these mathematical functions and all mathematical operators in C^H can be changed, added, or removed at user's convenience as delineated in(Cheng, 1993a).

4.1. Results of Dual Functions with Dual Numbers

The built-in polymorphic mathematical functions related to the dual numbers are listed in Table 8 along with their definitions. The input arguments of these dual functions can be dual numbers, dual variables, or dual expressions. For presentation purpose, the dual numbers d, d_1, and d_2 are defined as $x + \varepsilon y, x_1 + i\varepsilon y_1$, and $x_2 + \varepsilon y_2$, respectively. The results of dual functions involving dual metanumbers will be discussed in the next section.

The first four functions in Table 8 return real numbers. The **sizeof**() function is compatible with the ANSI C. It returns, in bytes, an integer of the variable, type-specifier, or expression that it precedes. If the argument is a dual, it will return the value of 8 which is the number of bytes required for storing two floats of real and imaginary parts of a dual number. The **abs**(d) function computes the magnitude of a dual number. The returned data type is float. Function **abs**() essentially treats a dual number as a complex number. The functions **real**(d) and **imaginary**(d) returns the real and imaginary parts of a dual number, respectively. The results of **real**(d) and **imaginary**(d) are always floats. If the data type of the argument for **real**() is lower or equal to double, the input data will be cast into a float. If the data type of the argument for **imaginary**() is lower than or equal to double, the value of zero will be returned. The sign of a zero will be ignored in **real**(z) and **imaginary**(z) functions. For example, **real**(dual(−0.0,0.0)) will return 0.0. A dual number can be constructed from two real numbers by the dual construction function **dual**(x,y). If the input argument are not floats, they will be cast into floats according to the internal data conversion rules. The **conjugate**(d) function returns the dual conjugate $\overline{d}$ of the dual number d.

The function **polar**() is implemented mainly for the convenience of transformation between Cartesian and polar coordinates of a point represented by a complex or dual number. If there is only one input argument, a dual number

Table 9: Results of dual functions for DualZero, DualInf, and DualNaN.

function	d value and results		
	DualZero	DualInf	DualNaN
sizeof(d)	8	8	8
abs(d)	0.0	Inf	NaN
real(d)	0.0	NaN	NaN
imaginary(d)	0.0	NaN	NaN
conjugate(d)	DualZero	DualInf	DualNaN
polar(d)	ComplexZero	ComplexInf	ComplexNaN
sqrt(d)	DualZero	DualNaN	DualNaN
	sqrt(dual(0.0, y)) = DualInf, $y \neq 0.0$;		
	sqrt(dual(x, y)) = DualNaN, $x < 0.0$;		
exp(d)	dual(1.0, 0.0)	DualNaN	DualNaN
log(d)	DualInf	DualNaN	DualNaN
	log(dual(x,y)) = DualNaN, $x < 0.0$;		
	log(dual(0.0,y)) = DualInf, $y \neq 0.0$;		
log10(d)	DualInf	DualNaN	DualNaN
	log10(dual(x,y)) = DualNaN, $x < 0.0$;		
	log10(dual(0.0,y)) = DualInf, $y \neq 0.0$;		
sin(d)	DualZero	DualNaN	DualNaN
cos(d)	dual(1.0, 0.0)	DualNaN	DualNaN
tan(d)	DualZero	DualNaN	DualNaN
	tan(dual($\pi/2 + k * \pi, y$)) = DualInf		
asin(d)	DualZero	DualNaN	DualNaN
	asin(dual(±1.0, 0.0)) = $\pm pi/2$		
	asin(dual(±1.0, y)) = DualInf, $y \neq 0.0$		
	asin(dual(x, y)) = DualNaN, $\lvert x\rvert > 1.0$		
acos(d)	dual(pi/2, 0.0)	DualNaN	DualNaN
	acos(dual(1.0, 0.0)) = 0.0		
	acos(dual(−1.0, 0.0)) = pi		
	acos(dual(±1.0, y)) = DualInf, $y \neq 0.0$		
	acos(dual(x, y)) = DualNaN, $\lvert x\rvert > 1.0$		
atan(d)	DualZero	DualNaN	DualNaN
sinh(d)	DualZero	DualNaN	DualNaN
cosh(d)	dual(1.0, 0.0)	DualNaN	DualNaN
tanh(d)	DualZero	DualNaN	DualNaN
asinh(d)	DualZero	DualNaN	DualNaN
acosh(d)	dual(0.0, pi/2)	DualNaN	DualNaN
	acosh(dual(1.0, 0.0)) = DualZero		
	acosh(dual(1.0, y)) = DualInf, $y \neq 0.0$		
	acosh(dual(x, y)) = DualNaN, $x < 1.0$		
atanh(d)	DualNaN	DualNaN	DualNaN
	atanh(dual(±1.0, y)) = DualInf;		
	atanh(dual(x, y)) = DualNaN, $\lvert x\rvert > 1.0$;		
ceil(d)	DualZero	DualInf	DualNaN
floor(d)	DualZero	DualInf	DualNaN
ldexp(z, d2)	DualZero	DualInf	DualNaN
modf(z, &d2)	DualZero	DualZero	DualNaN
d2	DualZero	DualInf	DualNaN
frexp(z, &d2)	DualZero	DualInf	DualNaN
d2	DualZero	DualZero	DualNaN

will be trated as a complex number internally. The returned real and imaginary parts contains the magnitude and principal phase angle Θ, respectively, of the converted complex number from the input dual number. If there are two input arguments, a complex number z in the polar form will be returned. The first and second input arguments are the magnitude and argument of z, respectively. Details about the **polar**() function related to complex numbers are described in (Cheng, 1993b). The **sqrt**(d) function computes the square root of the dual number d. The **exp**(d) function computes the exponential function of the dual number d. The **log**(d) function computes the natural logarithm of the dual number d. For the convenience, the function **log10**(d) calculates the base-ten logarithm of the dual number d. The exponential function with a dual base can be calculated by the function **pow**(). If any one of two arguments of **pow**(d1, d2) is dual, the result is a dual. The definition for **pow**(d1, d2) given in Table 8 is obtained by the formula (6). This definition is equivalent to the definition of $\mathbf{pow}(d_1, d_2) = \mathbf{exp}(d2 * \mathbf{log}(d1))$ formally. But, there is a subtle difference for computer implementation. According to the definition given in Table 8, if the input dual arguments have identically zero imaginary parts, the result is a dual number with an identically zero imaginary part, i.e., **pow**(dual(x_1, 0.0), dual(x_2, 0.0)) = dual($x_1^{x_2}$, 0.0). Depending on the implementation, the imaginary part of **exp**(dual(x_2, 0.0)***log**(dual(x_1, 0.0))) may not be an identically zero due to roundoff errors.

The dual trigonometric functions **sin**(d), **cos**(d), and **tan**(d) compute the sine, cosine,and tangent of the dual value d, respectively. The dual inverse trigonometric functions **asin**(d), **acos**(d), and **atan**(d) return the principal values of the arc sine, arc cosine, and arc tangent of the dual value d, respectively. The dual parts of these dual inverse trigonometric functions are unique. The principal values of the real parts of the functions **asin**(d), **acos**(d), and **atan**(d) are in the ranges of $[-\pi/2, \pi/2]$, $[0, \pi]$, and $[-\pi/2, \pi/2]$, respectively. The dual hyperbolic functions **sinh**(d), **cosh**(d), and **tanh**(d) compute the hyperbolic sine, cosine,and tangent of the dual value d, respectively. The dual inverse hyperbolic functions **asinh**(d), **acosh**(d), and **atanh**(d) compute the principal values of the hyperbolic arc sine, cosine, and tangent of the dual value d, respectively. Functions **asinh**(d) and **atanh**(d) have unique values. The real part of the returned dual number of the function **acosh**(d) gives the principal value in the range of $[1.0, \mathrm{Inf}]$.

For functions **ceil**(d), **floor**(d), and **ldexp**(d1, d2), the real and imaginary parts are treated as if they were two separate real functions. The **ceil**(d) function returns a dual whose real and dual parts are the smallest integral values not less than the real and dual parts of d, respectively. The **floor**(d) function returns a dual whose real and dual parts are the largest integral values not greater than the real and dual parts of d, respectively. The **ldexp**(di1, d2) functions multiplies the real part of $d1$ by the real part of $d2$ raised to the power of 2, and the dual part of $d1$ by the dual part of $d2$ raised to the power of 2. The **fmod**(d1,d2) function computes the dual remainder of d1/d2. The functions **modf**(d1, dptr) and **frexp**(d1, dptr) are handled in the same manner. For these two functions, when the data type of the first arguments are dual, the data type of the second argument must be a pointer to dual. The **modf**(d1, dptr) function breaks the real part $x1$ of $d1$ into integer and fraction parts, each of

Table 10: Results of function pow(d1, d2) for DualZero, DualInf, and DualNaN.

pow(d1, d2)				
d1 value	d2 value			
	DualZero	d2	DualInf	DualNaN
DualZero	DualNaN	DualNaN	DualNaN	DualNaN
d1	dual(1.0, 0.0)	$d_1^{d_2}$	DualNaN	DualNaN
d1; x1 ≤ 0	DualNaN	DualNaN	DualNaN	DualNaN
DualInf	DualNaN	DualInf	DualNaN	DualNaN
DualNaN	DualNaN	DualNaN	DualNaN	DualNaN

Table 11: Results of function fmod(d1, d2) for DualZero, DualInf, and DualNaN.

fmod(d1, d2)				
d1 value	d2 value			
	DualZero	d2	DualInf	DualNaN
DualZero	DualNaN	DualZero	DualZero	DualNaN
d1	DualNaN	fmod(d1,d2)	d1	DualNaN
DualInf	DualNaN	DualNaN	DualNaN	DualNaN
DualNaN	DualNaN	DualNaN	DualNaN	DualNaN

which keeps the same sign of $x1$. The integer part is stored as a float in the object pointed to by the second argument *dptr*. The dual part of $d1$ is treated in the same manner, the integer part is stored at four bytes off the memory pointed to by the second argument *dptr*. The obtained fractional parts from real and dual components of $d1$ are returned as a dual. The function **frexp**(d1, dptr) breaks the real part $x1$ of $d1$ into a normalized fraction and an integral power of 2. The integer part is stored as a float in the object pointed to by the second argument *dptr*. The dual part of $d1$ is treated in the same manner, the integer part is stored at four bytes off the memory pointed to by the second argument *dptr*. The obtained fractional parts from real and dual components of $d1$ are returned as a dual.

4.2. Results of Dual Functions with Dual Metanumbers

Like dual arithmetic operations, the definition for dual functions may not be valid when the input arguments are dual metanumbers. The results of the built-in dual functions with dual metanumbers as their input arguments are given in Tables 9-12.

In Table 9, dual($\pm 0.0, \pm 0.0$) in C$^{\mathrm{H}}$ is treated as dual(0.0, 0.0). When the input argument of a function is a Dual-Not-a-Number, the returned result is always DualNaN except for the function **sizeof**(). As shown in Figure 2, a dual infinity is different from the real infinities of $\pm\infty$. When either real or dual part of a dual value is outside the range of the representable floating-point number, it becomes DualInf. Therefore, the absolute value of DualInf is a real number of Inf. The real and imaginary parts of DualInf are indeterminate. But, the conjugate of DualInf is still a dual infinity. The result **polar**(dual(0.0,0.0)) is defined as complex(0.0,0.0) because **atan2**(0.0,0.0) equals 0.0 as described in (Cheng, 1993a). The result of **polar**(DualInf) is defined as ComplexInf.

Being consistent with the real function, the square root of the dual number of dual(0.0,0.0) is dual(0.0,0.0). The square root of a pure dual is DualInf. The square root of a dual number with negative real part is undefined. Both $-$Inf and Inf are treated as DualInf, therefore, the **sqrt**(DualInf) is DualNaN. As a real function, **exp**(Inf) = Inf whereas **exp**($-$Inf) = 0.0. Both values of $\pm$Inf become DualInf if they are cast into dual numbers. Therefore, the dual exponential function **exp**(d) is DualNaN when the input argument is DualInf. The dual logarithmic function **log**(d) with the input argument of dual(0.0,0.0) or dual(0.0, y) will return DualInf. When the real part of a dual number is negative, its logarithm is undefined. Hence, for the same reason of **sqrt**(DualInf), **log**(DualInf) is DualNaN.

The dual trigonometric functions **sin**(d), **cos**(d), and **tan**(d); and dual inverse trigonometric functions **asin**(d), **acos**(d), and **atan**(d) are undefined when the input arguments are DualInfs. The irrational number π is not representable in a computer program. If we had the value of π, the expression of **tan**($k\pi + \pi/2$) would return DualInf. If the real part of the dual number d is larger than 1.0, its arc sine and cosine are undefined. The values at d = dual($\pm 1.0, y$) for **asin**(d) and **acos**(d) are specially defined for compatibility with real functions. The results of **asin**(dual($\pm 1.0, y$)) and **acos**(dual($\pm 1.0, y$)) are DualInf if $y \neq 0.0$.

The dual hyperbolic functions **sinh**(d), **cosh**(d), and **tanh**(d); and dual inverse hyperbolic functions **asinh**(d), **acosh**(d), and **atanh**(d) are undefined if d = DualInf. The function **acosh**(d) is undefined when the real part x of d is less than 1.0. For compatibility with real functions, **acosh**(dual(1.0,0) is defined as dual(0.0,0.0) and **acosh**(dual(1.0,y) with $y \neq 0.0$ as DualInf. The result of **atanh**(dual(x,y)) is undefined if $|x| > 1.0$. The result of **atanh**(dual($\pm 1.0, y$)) is defined as DualInf.

With dual metanumbers as their input arguments, the real and dual parts of functions **ceil**(d), **floor**(d), and **ldexp**(d1, d2) are handled as if they were two separate real functions.

Results of function **pow**(d_1, d_2) for dual metanumbers are given in Table 10. To simplify the matter, if the real part of d_1 is less than or equal to zero, **pow**(d_1, d_2) is undefined. Because both Inf and $-$Inf are considered as DualInf, if any input argument is DualInf, **pow**(d_1, d_2) is DualNaN. Results of function **fmod**(d1,d2) for dual metanumbers are given in Table 11. The results of the dual construction function **dual**(x,y) are given in Table 12. To construct a dual number, if either its real or dual part is NaN, the result is a Dual-Not-a-Number. Likewise, if either one of them is a value of $\pm\infty$, the result is DualInf. Normally, through dual arithmetic and dual functions, one shall not get a dual number with its real or dual part being the value of $-$Inf, Inf, or NaN while the other part is a regular real number. This kind of results can be obtained only explicitly through lvalues, which will be discussed in the next section.

5. Conclusions

Dual numbers are generalization of real numbers. Similar to the extended complex plane, for the first time, the extended dual plane is introduced in this paper. For scientific computing with dual numbers, the extended finite dual plane along with dual metanumbers of DualZero, DualInf, and DualNaN are also introduced in this paper. The dual arithmetic and relational operations, and dual functions are defined in the syntax of the C$^{\mathrm{H}}$ programming language. The results of dual operations and dual functions with dual metanumbers may differ from those obtained directly according to formal definitions for dual numbers. Due to polymorphism, the algorithms and resultant data types of arithmetic operations

Table 12: Results of function dual(x, y) for $0.0, \pm\infty$, and NaN.

dual(x, y)						
x value	y value					
	−Inf	−y1	0.0	y2	Inf	NaN
Inf	DualInf	DualInf	DualInf	DualInf	DualInf	DualNaN
x2	DualInf	dual(x2, −y1)	dual(x2, 0.0)	dual(x2, y2)	DualInf	DualNaN
0.0	DualInf	dual(0.0, −y1)	dual(0.0, 0.0)	dual(0.0, y2)	DualInf	DualNaN
−x1	DualInf	dual(−x1, −y1)	dual(−x1, 0.0)	dual(−x1, y2)	DualInf	DualNaN
−Inf	DualInf	DualInf	DualInf	DualInf	DualInf	DualNaN
NaN	DualNaN	DualNaN	DualNaN	DualNaN	DualNaN	DualNaN

depend on the data types of operands, and the algorithms and resultant data types of mathematical functions are related to the data types of input arguments. Since dual is implemented as a basic data type in C^H, the numerical computation can be handled in a much integrated fashion. Dual numbers are manipulated in the same manner as real or complex numbers in C^H. It is the first time, I believe, that such a simplicity is introduced in a general-purpose computer programming language for scientific computing with dual numbers. Sample C^H programs for computations of the motion screw of a rigid body displacement and forward kinematics of robot manipulators , which illustrate the handling of user's dual functions and dual arrays, respectively, can be found in (Cheng, 1993). Through displacement analysis of a RCCC mechanism, Cheng (1993c) also demonstrated the simplicity of programming with dual formulas in the C^H programming language. It is expected that, with conciseness, clarity, and programming simplicity of dual formulas, the recognition of dual numbers as a powerful mathematical tool, which is well-known to kinematicians, will reach wide audience in engineering and science.

6. Acknowledgement

The author would like to thank Prof. A. T. Yang and Prof. B. Ravani for reading drafts of this paper, their comments and, in general, their help and encouragement.

7. References

1. ANSI, *ANSI Standard X3.9-1978, Programming Language Fortran*, (revision of ANSI X2.9-1966), American National Standards Institute, Inc., NY, 1978.
2. ANSI, *ANSI Standard X3.159-1989, Programming Language C*, ANSI, Inc., NY, 1989.
3. Bottema, O. and Roth, B.: *Theoretical Kinematics*, North-Holland Publishing Comp., Amsterdam, 1979.
4. Cheng, H. H., Scientific Computing in the C^H Programming Language, Journal of *Scientific Programming*, Vol. 2, No. 3, Fall, 1993, pp. 49-75.
5. Cheng, H. H., Handling of Complex Numbers in the C^H Programming Language, Journal of *Scientific Programming*, Vol. 2, No. 3, Fall, 1993, pp. 76-106.
6. Cheng, H. H., Computations of Dual Numbers in the Extended Finite Dual Plane with Applications in Mechanisms, Department of MAME, UC Davis, Technical Report TR-MAME-93-103, March 2, 1993c.
7. Clifford, W. K., Preliminary Sketch of Bi-quaternions, *Proceedings of London Mathematical Society*, Vol. 4, 1873, pp. 361-395.
8. Denavit, J., Displacement Analysis of Mechanisms Based on (2x2) Matrices of Dual Numbers, VDI-Berichte, Vol. 29, 1958, pp. 81-89.
9. Dimentberg, F. M., The Determination of the Positions of Spatial Mechanisms, *Izdatel'stvo Akademii Nauk*, Moscow, USSR, 1950.
10. Dooley, J. R. and McCarthy, J. M., Spatial Rigid Body Dynamics Using Dual Quaternion Components, *Proc. of IEEE International Conf. on Robotics and Automation*, vol. 1, Sacramento, CA, April 1991, pp.90-95.
11. Guggenheimer, H. W., *Differential Geometry*, McGraw-Hill Book Co., NY, 1963.
12. Kernighan, B. W. and Ritchie, D. M., *The C Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, NJ, first edition, 1978; second edition, 1988.
13. Kotelnikov, A. P., Screw Calculus and Some of its Applications to Geometry and Mechanics, Annals of Imperial University of Kazan, 1895.
14. McAulay, A., *Octonions– A Development of Clifford's Bi-quaternions*, Cambridge University Press, 1898.
15. Pennock, G. R. and Yang, A. T., Application of Dual-Number Matrices to the Inverse Kinematics Problem of Robot Manipulators, *Trans. ASME, J. of Mechanisms, Transmissions, and Automation in Design*, Vol.107, June 1985, pp. 201-208.
16. Pennock, G. R. and Yang, A. T., Dynamic Analysis of Multi-Rigid-Body Open-Chain System, *Trans. ASME, J. of Mechanisms, Transmissions, and Automation in Design*, Vol.105, March 1983, pp. 28-34.
17. Ravani, B. and Roth B., Mappings of Spatial Kinematics, *Trans. ASME, J. of Mechanisms, Transmissions, and Automation in Design*, Vol.106, Sept. 1984, pp. 341-347.
18. Ravani, B. and Ge Q. J., Kinematic Localization for World Model Calibration in Off-Line Robot Programming Using Clifford Algebra, *Proc. of IEEE International Conf. on Robotics and Automation*, vol. 1, Sacramento, CA, April 1991, pp. 584-589.
19. Study, E., *Geometrie der Dynamen*, Leipzig, 1903.
20. Yang, A. T., *Application of Quaternion Algebra and Dual Numbers to the Analysis of Spatial Mechanisms*, Doctoral Dissertation, Columbia University, New York, 1963.
21. Yang, A. T. and Freudenstein, F., *Application of Dual-Number Quaternion Algebra to the Analysis of Spatial Mechanisms, Trans. ASME, J. of Applied Mechanics*, Vol. 86, No. 2, June 1964, pp. 300-308.