Analysis of the GCR method with mixed precision arithmetic using QuPAT

Tsubasa Saito[a,*], Emiko Ishiwata[b], Hidehiko Hasegawa[c]

[a] *Graduate School of Science, Tokyo University of Science, 1-3 Kagurazaka, Shinjuku-ku, Tokyo 162-8601, Japan*
[b] *Tokyo University of Science, 1-3 Kagurazaka, Shinjuku-ku, Tokyo 162-8601, Japan*
[c] *University of Tsukuba, 1-2 Kasuga, Tsukuba-shi, Ibaraki 305-8550, Japan*

Abstract

To verify computation results of double precision arithmetic, a high precision arithmetic environment is needed. However, it is difficult to use high precision arithmetic in ordinary computing environments without any special hardware or libraries. Hence, we designed the quadruple precision arithmetic environment QuPAT on Scilab to satisfy the following requirements: (i) to enable programs to be written simply using quadruple precision arithmetic; (ii) to enable the use of both double and quadruple precision arithmetic at the same time; (iii) to be independent of any hardware and operating systems.
To confirm the effectiveness of QuPAT, we applied the GCR method for ill-conditioned matrices and focused on the scalar parameters α and β in GCR, partially using DD arithmetic. We found that the use of DD arithmetic only for β leads to almost the same results as when DD arithmetic is used for all computations. We conclude that QuPAT is an excellent interactive tool for using double precision and DD arithmetic at the same time.

Keywords: quadruple precision arithmetic, mixed precision, the GCR method

1. Introduction

Floating point arithmetic operations governed by IEEE754, for example, double precision arithmetic, is used for calculation on conventional computers. How-

[*]Corresponding author
Email addresses: j1411702@ed.kagu.tus.ac.jp (Tsubasa Saito), ishiwata@rs.kagu.tus.ac.jp (Emiko Ishiwata), hasegawa@slis.tsukuba.ac.jp (Hidehiko Hasegawa)

ever, in floating point arithmetic, we cannot avoid the cancellation of significant digits, round-off errors and information loss. The iterative method for solving a system of linear equations converges in theory, but it is known that it may not converge in practice when floating point arithmetic is used. To examine the effect of these errors, we need to use higher than double precision arithmetic. However, it is difficult to use high precision arithmetic in ordinary computing environments without any special hardware. In addition, many software packages do not have a specification for high precision arithmetic.

Double-Double(DD) arithmetic uses numbers that are represented by two double precision floating point numbers to facilitate high precision arithmetic. DD arithmetic has been proposed for quasi quadruple precision arithmetic by Bailey [1] and it is based on the algorithm for error-free floating point arithmetic by Dekker [3] and Knuth [5]. QD [1] and Lis [6] are implementations of DD arithmetic as program libraries in C and C++. However, to use quadruple precision arithmetic, programs have to be rewritten and such rewritten programs cannot be executed without special hardware or a special library; also we meet debugging difficulties in the rewriting process.

We discuss the convenient quadruple precision arithmetic environment QuPAT (Quadruple Precision Arithmetic Toolbox) [8, 10] with DD arithmetic. In designing QuPAT, we focused on achieving the following goals: (i) the capacity to write programs simply to use quadruple precision arithmetic; (ii) simultaneous use of both double and quadruple precision arithmetic; (iii) independence from any particular hardware and operating systems. To satisfy these conditions, we chose the interactive numerical software Scilab.

To confirm the effectiveness of QuPAT, we attempted to improve the convergence of iterative methods by using high precision arithmetic for ill-conditioned matrices. We examined whether convergence of the Generalized Conjugate Residual (GCR) method could be improved with partial use of DD arithmetic. We retained a double precision matrix, and computed the matrix-vector product with double precision arithmetic. The scalar parameters α and β in GCR were calculated using DD.

This paper is organized as follows. In Section 2, we present DD arithmetic and describe the characteristics of QuPAT. In Section 3, we show the effectiveness of QuPAT for analyzing the convergence of the GCR method for a system of linear equations. In Section 4, we present a summary and discuss future work.

2. Implementation of QuPAT

In this section, we first explain the concept of DD arithmetic, which is based on error-free floating-point algorithms; then we describe how to implement QuPAT and some of its features.

2.1. DD arithmetic

A DD number is represented using two double precision floating point numbers. A real number α is represented as a DD number $A = (Ahi, Alo)$, as defined below:

$$Ahi = (\ \alpha \text{ rounded to a double precision number})$$
$$Alo = (\ (\alpha - Ahi) \text{ rounded to a double precision number})$$

The sign and the exponent part of a DD number depend only on *Ahi*. The four arithmetic operations of DD are defined only using double precision arithmetic and some error-free floating point arithmetic algorithms by Dekker[3] and Knuth[5]. Please see Saito et al. [8] and Bailey [1] for details of DD arithmetic.

2.2. Implementation of the DD arithmetic environment QuPAT

In [8], we implemented an environment for quadruple precision arithmetic QuPAT [10] using DD arithmetic. In designing QuPAT, we focused on the following points:

- Programs can be written simply to use quadruple precision arithmetic.
- Both double and quadruple precision arithmetic should be usable at the same time.
- Any hardware and operating systems should be able to be used.

To satisfy these conditions, we chose the popular interactive numerical software package Scilab [9]. Scilab offers the following features:

- New data types can be defined.
- Operator overloading can occur and can be used deliberately.

We now describe how QuPAT is implemented and which characteristics of Scilab are used for this implementation. As mentioned above, new data types can be defined in Scilab. We treat a data type using a combination of some data as a class in C++. In Scilab, double precision numbers are defined by the data type named ‘constant’. Scalars, vectors and matrices are treated in the same way as the data type ‘constant’. Therefore, by using the Scilab function ‘tlist’, we can define a new data type named ‘dd’ to contain DD numbers. In addition, defining only the data type ‘dd’ enables elements to be expanded into DD numbers naturally. Furthermore, we can make use of operator overloading for the new data type ‘dd’. Allowing operator overloading to compute a value of ‘dd’, we can use the same operators $(+, -, *, /)$ for quadruple precision arithmetic as for double precision arithmetic.

Figure 1 shows the relationship between Scilab (left) and QuPAT (right). We can use both ‘constant’ (i.e. double precision) and ‘dd’ (i.e. quadruple precision) data types at the same time, with the same operators $(+, -, *, /)$. To convert a value between the types ‘constant’ and ‘dd’, functions ‘d2dd’ and ‘DD.hi’ are used. The Scilab environment can be naturally extended to use DD arithmetic and mixed precision arithmetic with QuPAT.

Thus a Scilab program with QuPAT can use DD arithmetic by means of a small change in the definition of data types. As a result, we can avoid introducing bugs and even if some do occur, debugging is easy. As we implement QuPAT using only Scilab functions, we can use QuPAT in any computational environment for which Scilab is installed. The ratio of computation time between double and DD depends only on the number of double precision operations for DD arithmetic, and this ratio is machine independent.

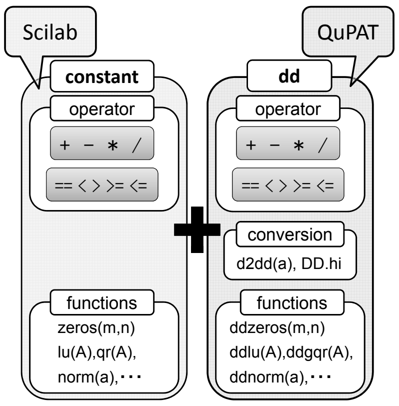

Figure 1: Relationship between Scilab and QuPAT in [10]

3. Analyzing the effect of round-off errors in the GCR method with QuPAT

The Generalized Conjugate Residual (GCR) method [4] is one of the Krylov subspace methods to solve a nonsymmetric linear system $A\boldsymbol{x} = \boldsymbol{b}$. GCR has the theoretical property that the residual norm of the approximate solution vector $||\boldsymbol{r}_k|| = ||\boldsymbol{b} - A\boldsymbol{x}_k||$ decreases at each iteration and converges after at most n iterations, where k is the number of iterations and n is the dimension of the matrix A. However, when using floating point arithmetic, sometimes the iteration process may not converge [2].

```
1.  Let $\boldsymbol{x}_0$ be an initial guess.
2.  set $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0,\ \boldsymbol{p}_0 = \boldsymbol{r}_0,\ \boldsymbol{q}_0 = A\boldsymbol{p}_0,\ k = 0$
3.  while $||\boldsymbol{r}_k||_2 < \varepsilon||\boldsymbol{r}_0||_2$ and $k < n$ do
4.      $\alpha_k = (\boldsymbol{r}_k, \boldsymbol{q}_k)/(\boldsymbol{q}_k, \boldsymbol{q}_k)$
5.      $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k$
6.      $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k\boldsymbol{q}_k$
7.      For $i = 0, \cdots, k$ do
8.          $\beta_{k,i} = -(A\boldsymbol{r}_{k+1}, \boldsymbol{q}_i)/(\boldsymbol{q}_i, \boldsymbol{q}_i)$
9.      $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \sum_{i=0}^{k} \beta_{k,i}\boldsymbol{p}_i$
10.     $\boldsymbol{q}_{k+1} = A\boldsymbol{r}_{k+1} + \sum_{i=0}^{k} \beta_{k,i}\boldsymbol{q}_i$
11.     $k = k + 1$
```

Figure 2: GCR algorithm [4]

Figure 2 shows the algorithm for GCR. In this section, we investigate the numerical solution of a linear system by GCR using double precision and DD arithmetic. All experiments were carried out on a PC with an AMD Turion X2 Dual-Core 2.0 GHz CPU and Scilab version 5.1.1 running on Windows XP. The iteration was started with $\boldsymbol{x}_0 = \boldsymbol{0}$ and the right-hand side vector $\boldsymbol{b}$ was given by substituting the solution $\boldsymbol{x}^* = (1, 1, ..., 1)^T$ into $\boldsymbol{b} = A\boldsymbol{x}^*$. Stopping criteria were $||\boldsymbol{r}_k||_2 \leq 10^{-12}\ ||\boldsymbol{r}_0||_2$ for double precision and $||\boldsymbol{r}_k||_2 \leq 10^{-14}\ ||\boldsymbol{r}_0||_2$ for DD. We chose a slightly stricter stopping criterion for DD compared to double precision.

3.1. Comparison between double and DD

We compare the results of using double precision arithmetic versus DD arithmetic. In this section, ‘double’ denotes the use of only double precision numbers and arithmetic and ‘DD’ denotes the use of only DD numbers and arithmetic. We compare the results of ‘double’ versus ‘DD’.

The difference in the program between ‘double’ and ‘DD’ is only in the definition of the variables and the name of the functions to compute a norm and an inner product in QuPAT. We take up DORR in Matlab and other test matrices from

MatrixMarket [7]. We choose $\theta = 0.004$ for DORR. Condition numbers are obtained using the Scilab function 'cond'. Table 1 shows the list of test matrices.

Table 1: Properties of test matrices

Matrix	Dimension	Condition number
arc130	130	6.05×10^{10}
impcol_c	137	1.77×10^{4}
fs_183_6	183	1.74×10^{11}
DORR ($\theta = 0.004$)	300	6.82×10^{14}
pores_3	532	5.61×10^{5}

Table 2: Comparison of the results of using double precision and DD arithmetic

Matrix	Double			DD		
	Iterations	Residual	Error	Iterations	Residual	Error
arc130	†	7.31e-11	1.82e+00	15	7.15e-16	2.40e-06
impcol_c	†	2.73e-01	2.97e+00	†	7.13e-10	3.01e-08
fs_183_6	†	8.40e-09	7.70e+00	45	7.25e-15	7.40e-06
DORR	114	9.08e-13	9.78e-01	155	8.49e-18	1.03e-08
pores_3	†	3.16e-06	3.06e-02	474	9.48e-15	3.81e-12

† : No convergence.

Table 2 shows the numerical results of 'double' and 'DD'. 'Iterations' denotes the number of iterations required for convergence, 'Residual' denotes $\|\boldsymbol{r}\|_2/\|\boldsymbol{r}_0\|_2$ and 'Error' denotes $\|\boldsymbol{x} - \boldsymbol{x}^*\|_\infty/\|\boldsymbol{x}^*\|_\infty$. We consider arc130 and give details of the comparison between 'double' and 'DD'. Figure 3 shows the convergence. In the case of 'double', the relative residual norm stagnated at about 1.0×10^{-10} and the error norm was 1.81×10^{0} after 10 iterations. On the other hand, in the case of 'DD', the relative residual norm became 7.15×10^{-16} and the error norm became 2.40×10^{-6} after 15 iterations. This is a significant improvement.

3.2. Comparison between α and β using mixed precision

The GCR for arc130 converges for 'DD' but does not converge for 'double'. This may be caused by round-off error. However, if we carry out all of the computation using DD, we need more time and storage space. For example, we can compare the computation time for 1 iteration by GCR for arc130. This takes 0.036 seconds with 'double', but 0.947 seconds with 'DD'; thus, DD takes about

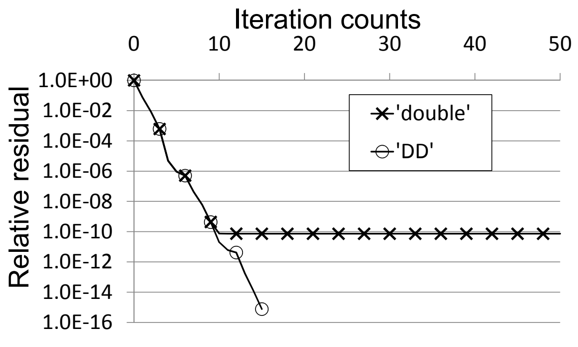

Figure 3: Convergence of arc130 in ‘double ’ and ‘DD’

26 times as long. In addition, a DD number requires twice the amount of storage. We investigated whether it was possible to improve convergence by using DD arithmetic only partially in the iterative process. We focused on two scalar parameters α_k and $\beta_{k,i}$ and considered the following two strategies:

（1） Compute the scalar α_k and vectors $\boldsymbol{x}_{k+1}$ and $\boldsymbol{r}_{k+1}$ with DD arithmetic from line 4 to line 6 in Figure 2. Retain α_k and $\boldsymbol{r}_{k+1}$ as DD numbers.

（2） Compute the scalar $\beta_{k,i}$ and vectors $\boldsymbol{p}_{k+1}$ and $\boldsymbol{q}_{k+1}$ with DD arithmetic from line 7 to line 10 in Figure 2. Retain $\beta_{k,i}$ and $\boldsymbol{q}_{k+1}$ as DD numbers.

Computation with 1 iteration of GCR for arc130 takes 0.078 seconds using strategy (1), and 0.370 seconds using strategy (2); thus the time is much less than for ‘DD’. Table 3 shows the numerical results of using strategies (1) and (2). The stopping criterion was $\|\boldsymbol{r}_k\|_2 \leq 10^{-14} \|\boldsymbol{r}_0\|_2$. With strategy (1), residual norms were not improved for all matrices. In contrast, with strategy (2), residual norms were improved for all matrices. GCR with strategy (2) converges for arc130, fs_183_6, DORR and pores_3. Note that the numbers of iterations for arc130, fs_183_6 and pores_3 are almost the same as when using ‘DD’. For impcol_c, strategy (2) resulted in a residual norm value of 3.25×10^{-8} and an error norm value of 2.40×10^{-6} after 137 iterations. This does not satisfy the stopping criterion, but is surely an improvement on the error norm value of 2.97×10^{0} of ‘double’. Figure 4 shows the convergence of DORR. The number of iterations of DORR using strategy (2) is 183, thus 18% more than is required by ‘DD’. The residual norm becomes 1.36×10^{-14}. However, the error norm is about 1.0×10^{-2}, which is not an improvement. Figure 5 shows the convergence of pores_3. The residual norm for pores_3 decreases gradually. In the case of ‘double’, the residual norm stagnates. However using strategy (2), the residual norm continues to decrease, and the convergence

Table 3: Partial use of DD arithmetic in GCR

Matrix	Strategy (1)			Strategy (2)		
	Iterations	Residual	Error	Iterations	Residual	Error
arc130	†	8.82e-11	2.29e+00	15	3.17e-15	2.40e-06
impcol_c	†	2.28e-01	2.92e+00	†	3.25e-08	2.40e-06
fs_183_6	†	8.84e-09	7.92e+00	47	7.18e-15	7.16e-06
DORR	†	8.31e-14	9.86e-01	183	1.36e-14	9.42e-03
pores_3	†	3.17e-06	3.07e-02	474	1.47e-14	4.05e-12

† : No convergence.

is similar to that of ‘DD’.

In this way, partially using DD arithmetic, such as with strategy (2), can improve the convergence of GCR. In some cases, using strategy (2) produces almost the same results as ‘DD’. The number of iterations is the same for both ‘DD’ and strategy (2). The computation time for arc130 is 8.27 seconds with ‘DD’ and 1.92 seconds for strategy (2) until GCR converges. We carried out the same experiment on another PC with an Intel Core2 Quad 2.83 GHz CPU and Scilab version 5.1.1 running on Windows XP. The computation time for arc130 is 2.20 seconds with ‘DD’ and 0.50 seconds for strategy (2). From the results of both PC environments, the computation time for strategy (2) is about 4 times faster than that with ‘DD’. Twice the storage space is required if all variables are declared as DD numbers. Using strategy (2), the required storage is about 1.5 times that required for ‘double’.

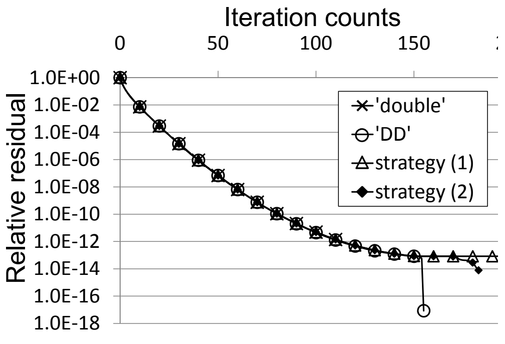

Figure 4: Convergence history for DORR

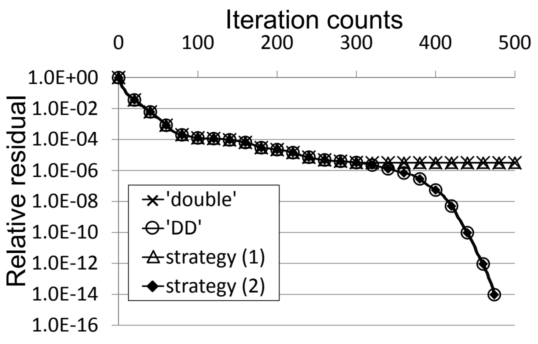

Figure 5: Convergence history for pores_3

3.3. *Further detailed examination for β*

In section 3.2, we confirmed that using strategy (2) improves the value of the residual norm. Recall that strategy (2) involves the computation of both the scalar $\beta_{k,i}$ and vectors $\boldsymbol{p}_{k+1}$ and $\boldsymbol{q}_{k+1}$ in DD arithmetic. We separate (2) into two sub-strategies to investigate further, as follows:

(2.1) Compute the scalar $\beta_{k,i}$ using DD arithmetic at line 8 in Figure 2. Retain $\beta_{k,i}$ as a DD number.

(2.2) Compute the vectors $\boldsymbol{p}_{k+1}$ and $\boldsymbol{q}_{k+1}$ with DD arithmetic at line 10 and line 11 in Figure 2. Retain $\boldsymbol{q}_{k+1}$ as a DD number.

Strategy (2) in Section 3.2 is thus a combination of (2.1) and (2.2). Table 4 shows the numerical results. The residual norms stagnated at almost the same rate as with 'double' for both strategies (2.1) and (2.2), and the residual norms were not improved. It is important to compute all of the following values with DD arithmetic:

- the inner product and the division for $\beta_{k,i}$ (line 8)
- $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \sum_{i=0}^{k} \beta_{k,i}\boldsymbol{p}_i$ (line 9)
- $\boldsymbol{q}_{k+1} = A\boldsymbol{r}_{k+1} + \sum_{i=0}^{k} \beta_{k,i}\boldsymbol{q}_i$ (line 10)

In addition, we need to declare $\boldsymbol{q}_{k+1}$ as a DD number. We are still able to use double precision arithmetic to compute the matrix-vector product $A\boldsymbol{r}_{k+1}$ at line 7 and line 10 in Figure 2.

Table 4: Further detailed examination of β

Matrix	Strategy (2.1)			Strategy (2.2)		
	Iterations	Residual	Error	Iterations	Residual	Error
arc130	†	1.02e-11	2.76e-01	†	6.41e-11	1.51e+00
impcol_c	†	2.23e-01	2.87e+00	†	2.69e-01	2.53e+00
fs_183_6	†	7.77e-10	4.62e+00	†	7.63e-09	8.33e+00
DORR	182	1.49e-14	9.22e-03	†	8.30e-14	9.83e-01
pores_3	†	2.51e-06	2.16e-02	†	3.18e-06	3.09e-02

† : No convergence.

3.4. *Result of partial accuracy improvement for GCR*

We investigate the improvement of convergence of the GCR method when DD arithmetic was partially used. Convergence was improved for five kinds of matrices by using strategy (2), which utilizes DD arithmetic for both computation and memory for the scalar $\beta_{k,i}$ and the vectors $\boldsymbol{p}_{k+1}$ and $\boldsymbol{q}_{k+1}$. In contrast, convergence was not improved by using DD arithmetic for the scalar $\beta_{k,i}$ alone or by using it for the vectors $\boldsymbol{p}_{k+1}$ and $\boldsymbol{q}_{k+1}$ alone. The computation time when using strategy (2) was nearly one quarter of that with 'DD'. Convergence was not improved by using strategy (1) that employed DD arithmetic for computing both the scalar α_k and the vectors $\boldsymbol{x}_{k+1}$ and $\boldsymbol{r}_{k+1}$. These strategies are much faster and use less memory than DD arithmetic requires. From these investigations, it appears that the round-off errors in computing the scalar $\beta_{k,i}$ and the vectors $\boldsymbol{p}_{k+1}$ and $\boldsymbol{q}_{k+1}$ may lead to the stagnation of the residual norm for ill-conditioned matrices in GCR.

To carry out these investigations, the only changes in the Scilab program for QuPAT involved the definition of data types and the name of the functions. In addition, we were able to use all of double, quadruple and mixed precision arithmetic in the same program with the same operators $(+, -, *, /)$. Therefore, we were able to easily compare the dependency of GCR on computing accuracy.

4. Conclusion

In this paper, we discussed an implementation of the interactive quadruple precision arithmetic environment QuPAT and showed the effectiveness of QuPAT by improving the convergence of the GCR method for ill-conditioned matrices. In designing QuPAT, we focused on these goals: (i) to enable programs to be written simply using quadruple precision arithmetic; (ii) to enable the use of both double and quadruple precision arithmetic at the same time; (iii) to be independent of any hardware and operating systems. To satisfy these conditions, we chose the interactive numerical software package Scilab. Using QuPAT, we can use double, quadruple and mixed precision arithmetic in the same program with the same operators $(+, -, *, /)$. Because of these features, we can write programs to use DD arithmetic in a simple fashion.

We investigated the improvement in convergence of the GCR method which arises from partially using DD arithmetic with QuPAT. We investigated accuracy improvements with respect to the two scalars α_k and $\beta_{i,k}$ in GCR. Convergence was improved by using DD arithmetic for computing both the scalar $\beta_{k,i}$ and the vectors $\boldsymbol{p}_{k+1}$ and $\boldsymbol{q}_{k+1}$. In contrast, convergence was not improved by using DD arithmetic for computing either the scalar $\beta_{k,i}$ or the vectors $\boldsymbol{p}_{k+1}$ and $\boldsymbol{q}_{k+1}$ independently. If we carry out all of the computation using DD arithmetic, we need more time and storage space. A DD number requires twice the amount of storage. Using our strategy, required storage is about 1.5 times that required for using only double precision arithmetic. In addition, computation time by our strategy is about one quarter compared to that where DD is used for all computation.

QuPAT is an excellent tool to allow the use of double, DD and mixed precision arithmetic naturally, and to investigate the effect on round-off errors with a partial accuracy improvement. The Scilab program with QuPAT is able to incorporate DD arithmetic with a small change in the definition of data types and function names. Therefore, the readability of programs with QuPAT is almost the same as that of ordinary Scilab programs. As QuPAT is independent of any hardware and operating systems, these programs are executable without any other change in the computing environment. As a consequence, we can easily carry out an investigation such as the one above with QuPAT.

On Scilab, we can also implement quad-double arithmetic in the same way as DD arithmetic in QuPAT. We also intend to analyze other iterative algorithms using QuPAT. These topics will be discussed elsewhere.

References

[1] D. H. Bailey, QD (C++ / Fortran-90 double-double and quad-double package), Available at http://crd.lbl.gov/~dhbailey/mpdist/

[2] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. Van der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition, SIAM (1994).

[3] T. J. Dekker, A Floating-Point Technique for Extending the Available Precision, Numer. Math. Vol. 18, pp. 224-242 (1971).

[4] S. C. Eisenstat, H. C. Elman and M. H. Schultz, Variational Iterative Methods for Nonsymmetric Systems of Linear Equations, SIAM J. Numer. Anal. Vol. 20, pp. 345-357 (1983).

[5] D. E. Knuth, The Art of Computer Programming, vol. 2. Addison Wesley (1969).

[6] Lis, http://www.ssisc.org/lis/

[7] MatrixMarket, http://math.nist.gov/MatrixMarket/

[8] T. Saito, E. Ishiwata and H. Hasegawa, Development of Quadruple Precision Arithmetic Toolbox QuPAT on Scilab, Proceeding of Computational Science and Its Applications - ICCSA2010, Part II, LNCS 6017, pp. 60-70, Springer-Verlag (2010).

[9] Scilab, http://www.scilab.org/

[10] QuPAT, http://www.mi.kagu.tus.ac.jp/qupat.html