Practical Implementation of High-Order Multiple Precision Fully Implicit Runge-Kutta Methods with Step Size Control Using Embedded Formula

Tomonori Kouya
Shizuoka Institute of Science and Technology
2200-2 Toyosawa, Fukuroi, Shizuoka 437-8555 Japan[*†]

May 11, 2013

Abstract

We propose a practical implementation of high-order fully implicit Runge-Kutta(IRK) methods in a multiple precision floating-point environment. Although implementations based on IRK methods in an IEEE754 double precision environment have been reported as RADAU5 developed by Hairer and SPARK3 developed by Jay, they support only 3-stage IRK families. More stages and higher-order IRK formulas must be adopted in order to decrease truncation errors, which become relatively larger than round-off errors in a multiple precision environment. We show that SPARK3 type reduction based on the so-called W-transformation is more effective than the RADAU5 type one for reduction in computational time of inner iteration of a high-order IRK process, and that the mixed precision iterative refinement method is very efficient in a multiple precision floating-point environment. Finally, we show that our implementation based on high-order IRK methods with embedded formulas can derive precise numerical solutions of some ordinary differential equations.

[*]2010 Mathematics Subject Classification: 65L06, 65F10, 65G50

[†]Keywords and phrases: Implicit Runge-Kutta Method, Multiple Precision Floating-point Arithmetic, Iterative Refinement Method

1 Introduction

Multiple precision floating-point (MP) arithmetic is an effective approach to solving ill-conditioned problems that cannot be solved precisely with IEEE754 double precison floating-point (DP) arithmetic. We have been developing BNCpack, a DP and MP numerical computation library based on MPFR[10], an arbitrary precision and IEEE754 standard compatible floating-point arithmetic library, for the natural number arithmetic kernel in GNU MP (GMP)[1] that is well-tuned for various CPU architectures. In this paper, we propose the implementation of a practical ordinary differential equation (ODE) solver based on BNCpack and high-order implicit Runge-Kutta (IRK) methods; its availability and efficiency are verified via numerical experiments.

DP ODE solvers based on IRK methods have been developed as RADAU5 (Radau IIA formula) and SPARK3 (selectable in Radau, Gauss, and Lobatto formulas) by Haier and Jay, respectively. Both IRK implementations support only 3-stage formulas, which is not sufficient to obtain precise numerical solutions in an MP environment. The use of MP floating-point arithmetic decreases round-off errors in an IRK process, thereby increasing truncation errors. More stages and higher-order IRK formulas are neccessary in an MP environment. Consequently, higher-dimensional nonlinear equations must be solved in high-order IRK processes. Such so-called inner iteration includes linear equations of the same dimension, which must be solved efficiently. In this process, RADAU5 reduces these coeffient matrices to complex diagonal matrices, and SPARK3 reduces them to real nonsymmetric tridiagonal matrices. In an MP environment, we must also accelerate these processes. In addition, we must be able to control the step sizes in an IRK process by using local error estimation at each discrete point in the given integration interval. For this reason, embedded formulas incidental to high-order IRK ones are neccessary. Althought Hairer proposed 4-stage embedded formula incidental to 3-stage Radau IIA formula, he did not describe explicitly the existence of embedded formulas for other IRK ones.

In this paper, we first state mathematical definitions and provide a framework for IRK algorithms. In section 3, we explain the method of linear equations to be solved in the inner iteration of an IRK process. We show that the RADAU5 type reduction is not effective for high-order IRK methods, and that the mixed precision iterative refinement method can achive drastic acceleration, as shown via benchmark tests of a linear ODE. In section 4, we describe the derivation of embedded formulas for any IRK ones, and we discuss the A-stabilities. In section 5, we describe numerical experiments conducted for some test problems in order to demonstrate the high performance of our implementations. Finally, we conclude this paper and discuss the scope for future studies in section 6.

2 Algorithm of Implicit Runge-Kutta Method

We define the initial value proble (IVP) of the ODE to be solved as

$$\begin{cases} \dfrac{d\mathbf{y}}{dx} = \mathbf{f}(x, \mathbf{y}) \in \mathbb{R}^n \\ \\ \mathbf{y}(x_0) = \mathbf{y}_0. \end{cases} \tag{1}$$

The integration interval is given as $[x_0, \alpha]$.

For this IVP of the ODE, we divide the integration interval into x_0, $x_1 := x_0 + h_0$, ..., $x_{k+1} := x_k + h_k$, ... In order to obtain numerical solutions $\mathbf{y}_k \approx \mathbf{y}(x_k)$ in each step by using an m-stage IRK method, we must solve the following nonlinear system of equations (*). This process of solving with various iterative methods called inner iteration in IRK methods.

$$(*) \begin{cases} \mathbf{k}_1 &= \mathbf{f}(x_k + c_1 h_k, \mathbf{y}_k + h_k \cdot \sum_{j=1}^m a_{1j}\mathbf{k}_j) \\ \mathbf{k}_2 &= \mathbf{f}(x_k + c_2 h_k, \mathbf{y}_k + h_k \cdot \sum_{j=1}^m a_{2j}\mathbf{k}_j) \\ &\vdots \\ \mathbf{k}_m &= \mathbf{f}(x_k + c_m h_k, \mathbf{y}_k + h_k \cdot \sum_{j=1}^m a_{mj}\mathbf{k}_j) \end{cases} \tag{2}$$

$$\mathbf{y}_{k+1} := \mathbf{y}_k + h_k \cdot \sum_{j=1}^{m} b_j \mathbf{k}_j$$

where the constant coefficients in the IRK formula, c_1, ..., c_m, a_{11}, ..., a_{mm}, b_1, ..., b_m, can be expressed as follows:

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1m} \\ c_2 & a_{21} & a_{21} & \cdots & a_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ c_m & a_{m1} & a_{m2} & \cdots & a_{m,m} \\ \hline & b_1 & b_2 & \cdots & b_m \end{array} = \begin{array}{c|c} \mathbf{c} & A \\ \hline & \mathbf{b}^T \end{array}.$$

All computations are the same in each discretization point x_k; hence, we consider only the computation $\mathbf{y}_0 \to \mathbf{y}_1 \approx \mathbf{y}(x_0 + h_0) = \mathbf{y}(x_0 + h)$.

Quasi-Newton Method If Newton method is adpoted as the numerical argorithm in inner iteration, the algorithm is as follows. The initial guesses are $\mathbf{k}_1^{(0)}$, ..., $\mathbf{k}_m^{(0)}$, and the approximations of unknowns $\mathbf{k}_1$, $\mathbf{k}_2$, ..., $\mathbf{k}_m$ are calculated by iterating the computations as

$$\begin{bmatrix} \mathbf{k}_1^{(l+1)} \\ \mathbf{k}_2^{(l+1)} \\ \vdots \\ \mathbf{k}_m^{(l+1)} \end{bmatrix} := \begin{bmatrix} \mathbf{k}_1^{(l)} \\ \mathbf{k}_2^{(l)} \\ \vdots \\ \mathbf{k}_m^{(l)} \end{bmatrix} - J^{-1}(\mathbf{k}_1^{(l)}, ..., \mathbf{k}_m^{(l)}) \begin{bmatrix} \mathbf{k}_1^{(l)} - \mathbf{f}(x_0 + c_1 h, \mathbf{y}_0 + h \sum_{j=1}^m a_{1j}\mathbf{k}_j^{(l)}) \\ \mathbf{k}_2^{(l)} - \mathbf{f}(x_0 + c_2 h, \mathbf{y}_0 + h \sum_{j=1}^m a_{2j}\mathbf{k}_j^{(l)}) \\ \vdots \\ \mathbf{k}_m^{(l)} - \mathbf{f}(x_0 + c_m h, \mathbf{y}_0 + h \sum_{j=1}^m a_{mj}\mathbf{k}_j^{(l)}) \end{bmatrix}$$

where $J(\mathbf{k}_1^{(l)}, \mathbf{k}_2^{(l)}, ..., \mathbf{k}_m^{(l)}) \in \mathbb{R}^{mn \times mn}$ is given by

$$J(\mathbf{k}_1^{(l)}, \mathbf{k}_2^{(l)}, ..., \mathbf{k}_m^{(l)}) = \left[\begin{array}{c|c|c|c} I_n - J_{11} & -J_{12} & \cdots & -J_{1m} \\ \hline -J_{21} & I_n - J_{22} & \cdots & -J_{2m} \\ \hline \vdots & \vdots & & \vdots \\ \hline -J_{m1} & -J_{m2} & \cdots & I_n - J_{mm} \end{array}\right],$$

I_n is an n-dimensional identity matrix, and J_{pq} is given by

$$J_{pq} = ha_{pq}\frac{\partial}{\partial \mathbf{y}}\mathbf{f}(x_0 + c_p h, \mathbf{y}_0 + h\sum_{j=1}^{m} a_{pj}\mathbf{k}_j^{(l)}) \in \mathbb{R}^{n \times n}.$$

In order to compute this part, we solve the following mn-dimensional system of linear equations with the coefficient matrix $J(\mathbf{k}_1^{(l)}, \mathbf{k}_2^{(l)}, ..., \mathbf{k}_m^{(l)})$ for unknowns $[\overline{\mathbf{z}_1}\ \overline{\mathbf{z}_2}\ ...\ \overline{\mathbf{z}_n}]^T$.

$$J(\mathbf{k}_1^{(l)}, \mathbf{k}_2^{(l)}, ..., \mathbf{k}_m^{(l)}) \begin{bmatrix} \overline{\mathbf{z}_1} \\ \overline{\mathbf{z}_2} \\ \vdots \\ \overline{\mathbf{z}_m} \end{bmatrix} = \begin{bmatrix} \mathbf{k}_1^{(l)} - \mathbf{f}(x_0 + c_1 h, \mathbf{y}_0 + h\sum_{j=1}^{m} a_{1j}\mathbf{k}_j^{(l)}) \\ \mathbf{k}_2^{(l)} - \mathbf{f}(x_0 + c_2 h, \mathbf{y}_0 + h\sum_{j=1}^{m} a_{2j}\mathbf{k}_j^{(l)}) \\ \vdots \\ \mathbf{k}_m^{(l)} - \mathbf{f}(x_0 + c_m h, \mathbf{y}_0 + h\sum_{j=1}^{m} a_{mj}\mathbf{k}_j^{(l)}) \end{bmatrix}$$

In this process, we need to have $O(m^2n^2)$ memory in order to store $J(\mathbf{k}_1^{(l)}, \mathbf{k}_2^{(l)}, ..., \mathbf{k}_m^{(l)})$.

Actually, to accelerate this process, the following fixed coefficient matrix J is used.

$$J(\mathbf{k}_1^{(l)}, \mathbf{k}_2^{(l)}, ..., \mathbf{k}_m^{(l)}) = J(\mathbf{k}_1^{(0)}, \mathbf{k}_2^{(0)}, ..., \mathbf{k}_m^{(0)})$$

We call this method "Quasi-Newton Method."

Simplified Newton Method For inner iteration in IRK methods, a more simplified Newton method is used conventionally[3]. RADAU5 and SPARK3 select the simplified Newton method.

In this process, J_{pq} used in the Quasi-Newton method is fixed as

$$J_{pq} := ha_{pq}\frac{\partial}{\partial \mathbf{y}}\mathbf{f}(x_0, \mathbf{y}_0) = ha_{pq}J,$$

and then, we use $Y_i = \mathbf{y}_0 + h\sum_{j=1}^{m} a_{ij}\mathbf{f}(x_0 + c_i h, Y_j)$, which is an alternative to $\mathbf{k}_i^{(l)}$. Thus, we can express the system of linear equations to be solved as

$$(I_m \otimes I_n - hA \otimes J)\mathbf{Z} = -\mathbf{F}(\mathbf{Y}) \in \mathbb{R}^{mn}, \tag{3}$$

where

$$\mathbf{F}(\mathbf{Y}) = \begin{bmatrix} Y_1 - \mathbf{y}_0 - h\sum_{j=1}^{m} a_{1j}\mathbf{f}(x_0 + c_1 h, Y_1) \\ \vdots \\ Y_m - \mathbf{y}_0 - h\sum_{j=1}^{m} a_{mj}\mathbf{f}(x_0 + c_m h, Y_m) \end{bmatrix}.$$

3 Acceleration of Inner Iteration in IRK process

In this section, we treat the simplified Newton method as inner iteration. In this case, the system of linear equations to be solved at each step of inner iteration is expressed in (3). We can decrease the number of computations by applying reductions based on similarity transformations to IRK matrix A. RADAU5 employs complex diagonalization, and SPARK3 employs real unsymmetric tridiagonalization, which is called W-transformation. The latter transformation is better than the former because it does not treat complex arithmetic, and it can avoid the ill-conditioned transformations. Therefore, we employ SPARK3 type reduction for the implementation of high-order IRK methods. In the rest of this section, we compare RADAU5 type and SPARK3 type reductions, and we show that the acceleration due to the mixed precision iterative refinement method can drastically reduce the computational time of an IRK process.

3.1 Comparison between RADAU5 type and SPARK3 type Reductions

RADAU5 type reduction[3] is based on the fact that IRK matrix A can be transformed into a complex diagonal matrix as

$$\Lambda = \mathrm{diag}(\lambda_1, \cdots, \lambda_m) = SAS^{-1}.$$

In these cases of IRK Radau, Gauss, and Lobatto formula families, eigenvalues λ_i are generally complex numbers.

If RADAU5 type reduction is employed in the simplified Newton method, the system of linear equations to be solved has the coefficient matrix obtained by similarity transformation with $S \otimes I_n$ and $S^{-1} \otimes I_n$, as

$$\begin{aligned} &(S \otimes I_n)(I_m \otimes I_n - hA \otimes J)(S^{-1} \otimes I_n) = I_m \otimes I_n - h\Lambda \otimes J \\ &= \begin{bmatrix} I_n - h\lambda_1 J & & \\ & \ddots & \\ & & I_n - h\lambda_m J \end{bmatrix}. \end{aligned}$$

RADAU5 type reduction has two advantages: 1. The inner iteration is completely parallelizable, and 2. the order of the required memory is $O(2mn)$. However, Λ, S, and S^{-1} are complex matrices, and $\kappa_2(S) = \|S\|_2 \|S^{-1}\|_2 \to \infty \ (m \to \infty)$; hence, round-off errors increase in inner iteration, especially for higher order IRK formulas, as shown in Figure 1.

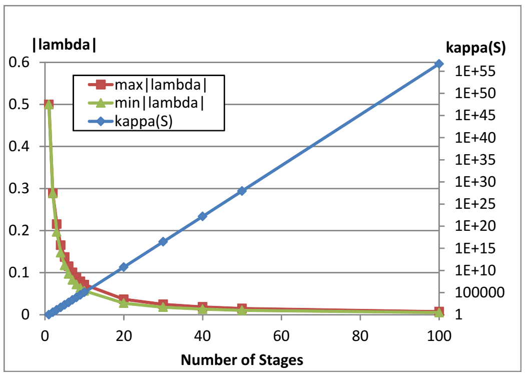

Figure 1: $\kappa_2(S)$ and absolute values of eigenvalues of A for IRK Gauss formulas

On the other hand, SPARK3 type reduction is called block triangulation through W-transformation[3], given by

$$X = W^T BAW = \begin{bmatrix} 1/2 & -\zeta_1 & & & \\ \zeta_1 & 0 & \ddots & & \\ & \ddots & \ddots & -\zeta_{m-2} & \\ & & \zeta_{m-2} & 0 & -\zeta_{m-1} \\ & & & \zeta_{m-1} & 0 \end{bmatrix},$$

where

$$w_{ij} = \tilde{P}_{j-1}(c_i) = \sqrt{2(j-1)+1} \sum_{k=1}^{j-1} (-1)^{j+k-1} \begin{pmatrix} j-1 \\ k \end{pmatrix} \begin{pmatrix} j+k-1 \\ k \end{pmatrix} c_i^k$$

$\tilde{P}_{j-1}(x)$: $j-1$-th shifted Legendre polynomial

$$\zeta_i = \left(2\sqrt{4i^2-1}\right)^{-1}, \ B = \mathrm{diag}(\mathbf{b}), \ D = W^T BW = \mathrm{diag}(1\ 1\ \cdots\ 1).$$

By using W-transformation, we can obtain the reduced coefficient matrix in the simplified Newton method [6] as

$$\begin{aligned} &(W^T B \otimes I_n)(I_m \otimes I_n - hA \otimes J)(W \otimes I_n) \\ &= D \otimes I_n - hX \otimes J = \begin{bmatrix} E_1 & F_1 & & & \\ G_1 & E_2 & F_2 & & \\ & \ddots & \ddots & \ddots & \\ & & G_{m-2} & E_{m-1} & F_{m-1} \\ & & & G_{m-1} & E_m \end{bmatrix}, \end{aligned}$$

Table 1: Condition numbers of two kinds of similarity transformation matrices

m	3	5	10	15	20	50
$\kappa_\infty(S)$	22.0	388	3.28×10^5	2.81×10^8	2.11×10^{11}	4.25×10^{28}
$\kappa_\infty(W)$	3.24	6.27	16.4	29.3	44.5	172

where

$$\begin{array}{l} E_1 = I_n - \frac{1}{2}hJ, \ E_2 = \cdots = E_s = I_n \\ F_i = h\zeta_i J, \ G_i = -h\zeta_i J \ (i = 1, 2, ..., m-1) \end{array}.$$

In the implementation of SPARK3, the left preconditioner matrix P,

$$P = \begin{bmatrix} \tilde{E}_1 & F_1 & & & \\ G_1 & \tilde{E}_2 & F_2 & & \\ & \ddots & \ddots & \ddots & \\ & & G_{m-2} & \tilde{E}_{m-1} & F_{m-1} \\ & & & G_{m-1} & \tilde{E}_m \end{bmatrix} \approx D \otimes I_n - hX \otimes J$$

is prepared, and then, the precondtioned system of linear equation

$$P^{-1}(D \otimes I_n - hX \otimes J)(W \otimes I_n)^{-1}\mathbf{Z} = P^{-1}(W^T B \otimes I_n)(-\mathbf{F}(\mathbf{Y}))$$

is set to be solved for $\mathbf{Z}$. Jay maintained that the left precondition can decrease the number of iterations and accelerate Richardson iteration and GMRES (Generalized Minimal RESidual) methods[6]. However, our numerical experiments show that such preconditioning increases the computational time in MP environment; hence, our current implementation employs such preconditions in DP environment, and does not employ in MP environment.

In comparison with RADAU5 type reduction, SPARK3 type reduction is better because all computations constitute real number arithmetic, and the order of the memory required for the coefficient matrix is $O(3mn)$. In addition, the similarity transformation matrix W can remain well-conditioned when the number of stages m is large, and hence, the effect of round-off errors occuring in the similarity transformation is small. Table 1 shows that the condition number of $\kappa_\infty(W) = ||W||_\infty ||W^{-1}||_\infty$ is much smaller than $\kappa_\infty(S)$ used in RADAU5 type reduction.

3.2 Acceleration by using Mixed Precision Iterative Refinement Method

The mixed precision iterative refinement method was originally proposed by Moler in 1967[9], and then Buttari et al. showed that their revised algorithm exhibits high performance in many current computing environments. If S-digit floating-point arithmetic can be executed more efficiently than L ($>> S$)-digit arithmetic, the system of linear equations

$$\mathbf{f}(\mathbf{x}) = C\mathbf{x} - \mathbf{d}$$

can be solved by Newton method and an appropriate linear solver, shown by the following algorithm:

```
Solve $C^{[S]}\mathbf{x}_0^{[S]} = \mathbf{d}^{[S]}$ for $\mathbf{x}_0^{[S]}$.        (4)
 $\mathbf{x}_0^{[L]} := \mathbf{x}_0^{[S]}$
 For $k = 0, 1, 2, \ldots$
     $\mathbf{r}_k^{[L]} := \mathbf{d}^{[L]} - C^{[L]}\mathbf{x}_k^{[L]}$        (5)
     $\mathbf{r}_k'^{[L]} := \mathbf{r}_k^{[L]}/||\mathbf{r}_k^{[L]}||$
     $\mathbf{r}_k'^{[S]} := \mathbf{r}_k'^{[L]}$
     Solve $C^{[S]}\mathbf{z}_k^{[S]} = \mathbf{r}_k'^{[S]}$ for $\mathbf{z}_k^{[S]}$.        (6)
     $\mathbf{z}_k^{[L]} := \mathbf{z}_k^{[S]}$
     $\mathbf{x}_{k+1}^{[L]} := \mathbf{x}_k^{[L]} + ||\mathbf{r}_k^{[L]}||\mathbf{z}_k^{[L]}$        (7)
     Check convergence of $\mathbf{x}_{k+1}$.
```

where $[S]$ and $[L]$ denote the values expressed and computed in S- and L-digit floating-point arithmetic, respectively. The above algorithm is the S-L mixed precision iterative refinement method for a system of linear equations. The part of (4) and (6) theoretically give the solution, and hence, we do not need the above iteration. However, we cannot obtain the true solutions owing to the use of finite precision floating-point arithmetic in these parts. Thus, the residuals $\mathbf{r}_k$ are not all zero. The above algorithm executes some iterations so that the residuals $\mathbf{r}_k$ tend to zero. The number of digits required to compute the residuals increases with the precision of the obtained approximation $\mathbf{x}_k$.

Buttari et. al. also proved that the sufficient condition for convergence is satisfied if the precision of the computation (6) is less than that of the residual (5) when the condition number $\kappa(A) = ||A||||A^{-1}||$ is smaller than the precision in the computing environment. In addition, they showed that the combination of (5) and (7) computed DP arithmetic, and (4) and (6) computed in SP(IEEE754 single-precision) arithmetic can accelerate the solution of relatively less ill-conditioned systems of linear equations via benthmark[2][7].

Their mixed precision iterative refinement method is just fitted to solve the system of linear equations (3) in inner iteration of IRK methods. This tends to be well-conditioned if the step size h_k becomes smaller, and hence, the sufficient condition of convergence is satisfied in almost cases in an IRK process. In particular, for MP arithmetic, the application to the DP-MP type mixed precision iterative refinement method in which (6) is computed by DP arithmetic can drastically accelerate the entire computational time of an IRK process. We support direct methods and Krylov subspace methods in our current implementation of mixed precision iterative refinement method.

In the rest of this section, we show the acceleration of an IRK method with DP-MP type mixed precision iterative refinement method via benchmark tests applied to a linear ODE.

Our test problem is constructed by using a real normal matrix R comprising uniform random numbers, and by its inverse matrix R^{-1}, as follows:

$$\begin{cases} \dfrac{d\mathbf{y}}{dx} = -(R \operatorname{diag}(n, n-1, ..., 1)\, R^{-1})\, \mathbf{y} \ \in \mathbb{R}^{128} \\ \mathbf{y}(0) = [1 \ ... \ 1]^T \end{cases}$$

Integration Interval:$[0, 20]$.

The precision of MP arithmetic is fixed at 50 decimal digits (167 bits). We compute $\mathbf{y}_1$ by using the m-stage $2m$ order Gauss formulas ($m = 3, 4, ..., 12$ and $h = 1/2$), and we compare the following 4 algorithms:

1. Quasi-Newton method with the DP-MP type mixed precision iterative refinement method based on the direct method and without reduction: “Iter.Ref-DM”

2. Simplified Newton method with the simple direct method and with SPARK3 type reduction: “W-Trans”

3. Simplified Newton method with the MP (25 decimal digits)-MP iterative refinement method with SPARK3 type reduction: “W-Iter.Ref-MM”

4. Simplified Newton method with the DP-MP method with SPARK3 type reduction: “W-Iter.Ref-DM”

All computations are executed on an Intel Core i7 920 + CentOS 5.4 x86_64 machine with gcc 4.1.2 + BNCpack 0.8 + MPFR 3.1.0/GMP 5.0.2. For convenient comparison, the maximum relative error in approximation $\mathbf{y}_1$ is expressed as a line graph in Figure 2. All 4 algorithms can obtain the same accuracy of $\mathbf{y}_1$.

Although the relative errors over 9 stages undergo less reduction because of the effect of round-off errors, all numerical results from 3- to 12-stages IRK Gauss formulas are precise.

The computational time increases with number of stages employed in the IRK formulas. The Quasi-Newton method without reduction in inner iteration is competitive with less than four stages, but is extremely slow with over 9 stages. On

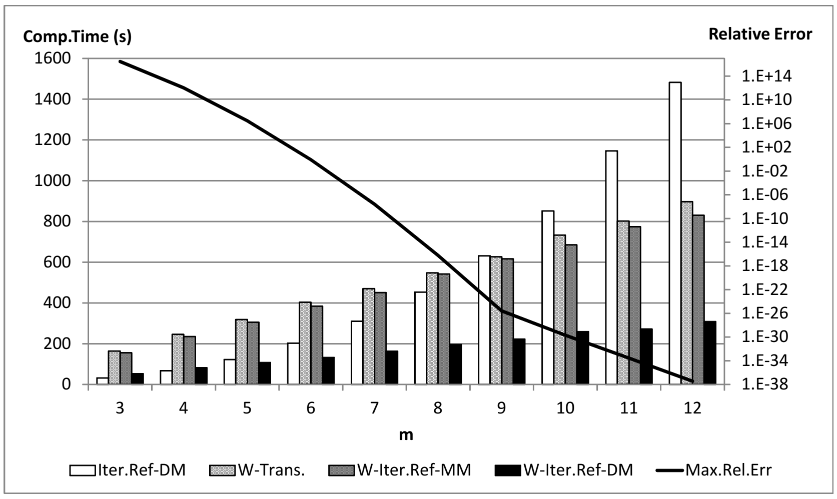

Figure 2: Performance of IRK methods: in case of 128-dimensional and 50 decimal digits computation

the other hand, the computational times of the three algorithms with SPARK3 type reduction is proportional to the number of stages. Moreover, the application of the DP-MP type iterative refinement method can drastically accelerate IRK processes, especially for formulas with over 5 stages formulas. The maximum obtained speedup ratio is 4.8.

4 Derivation of Embedded Formula and Step Size Selection

In order to implement actual ODE solvers, we need a mechanism of step size selection, based on the local error estimation at each discretized point. In the case of Runge-Kutta (RK) methods, embedded formulas that can reduce the amount of computation and the number of integrated function calls are widely used. For explicit RK (ERK) methods, many ODE solvers have been developed on the basis of the embedded formulas proposed by Fehlberg and Dormand-Prince. On the other hand, the embedded formula incident to 3-stage Radau IIA formula, which is proposed by Hairer[5] is only one for IRK methods, and it is constructed by a combination of the original formula and a lower order one. Hairer suggested the existence of the same type embedded formulas for other IRK formulas. In this section, we show how to automatically derive embedded formulas for IRK Gauss formulas which have wider A-stability regions.

4.1 Hairer's Embedded Formula

Hairer's embedded formula for the 3-stage 5th order Radau IIA formula is used in RADAU5, and it can be expressed as follows. The original Radau IIA is

$$\begin{array}{c|ccc} \frac{4-\sqrt{6}}{10} & \frac{88-7\sqrt{6}}{360} & \frac{296-169\sqrt{6}}{1800} & \frac{-2+3\sqrt{6}}{225} \\ \frac{4+\sqrt{6}}{10} & \frac{296+169\sqrt{6}}{1800} & \frac{88+7\sqrt{6}}{360} & \frac{-2-3\sqrt{6}}{225} \\ 1 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \\ \hline & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{array}$$

and its embedded formula is the 4-stage formula

$$\begin{array}{c|cc} c_0 = 0 & 0 & \mathbf{0}^T \\ \mathbf{c} & \mathbf{0} & A \\ \hline & \gamma_0 & \hat{\mathbf{b}}^T \end{array} \qquad (8)$$

where γ_0 is any non-zero constant, as recommended by Hairer for the real eigenvalue of IRK matrix A in order to reduce the number of computations in local error estimation. Moreover, $\hat{\mathbf{b}}$ is expressed as

$$\hat{\mathbf{b}} = \begin{bmatrix} \hat{b}_1 \\ \hat{b}_2 \\ \hat{b}_3 \end{bmatrix} = \begin{bmatrix} b_1 - \frac{2+3\sqrt{6}}{6}\gamma_0 \\ b_2 - \frac{2-3\sqrt{6}}{6}\gamma_0 \\ b_3 - \frac{\gamma_0}{3} \end{bmatrix}. \qquad (9)$$

This $\hat{\mathbf{b}}$ satisfies the simplifying assumption $B(3)$ [3]

$$B(3)\colon c_0^{q-1}\gamma_0 + \sum_{i=1}^{3} \hat{b}_i c_i^{q-1} = 1/q \ (q = 1, 2, 3).$$

The other coefficients $\mathbf{c}$ and A are the same as the original formula, and hence, the simplifying $C(3)$, given by

$$C(3)\colon \sum_{j=1}^{3} a_{ij} c_j^{q-1} = c_i^q / q \ (i = 1, 2, 3, q = 1, 2, 3),$$

is automatically satisfied. Therefore, the given embedded formula (8) is of 3rd order at least.

4.2 Derivation of Embedded Formulas for any IRK ones

Hairer's embedded formulation can be generally extended to other IRK formulas. In order to satisfy the simplifying assumption $B(m)$, $\hat{\mathbf{b}}$ is the solution of the following system of linear equations with a Vandermonde matrix of coefficients,

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ c_1 & c_2 & \cdots & c_m \\ \vdots & \vdots & & \vdots \\ c_1^{m-1} & c_2^{m-1} & \cdots & c_m^{m-1} \end{bmatrix} \begin{bmatrix} \hat{b}_1 \\ \hat{b}_2 \\ \vdots \\ \hat{b}_m \end{bmatrix} = \begin{bmatrix} 1-\gamma_0 \\ 1/2 \\ \vdots \\ 1/m \end{bmatrix}. \tag{10}$$

By solving the above equation, we can obtain the approximation $\hat{\mathbf{y}}_{k+1}$ derived by the m-th order embedded formula,

$$\hat{\mathbf{y}}_{k+1} := \mathbf{y}_k + h_k \left(\gamma_0 \mathbf{f}(x_k, \mathbf{y}_k) + \sum_{j=1}^{m} \hat{b}_j Y_j \right). \tag{11}$$

When we fix $\gamma_0 = 1/8$ for the reason described in the next section, we compute the relative errors of the approximation $\hat{y}_k$ in (12) at $x = 10$ by using the embedded formula derived from the 3-stage 6th order Gauss IRK formula.

$$\begin{aligned} &\frac{dy}{dx} = -xy \\ &y(0) = 1 \\ &\text{Integration Interval:}[0, 10] \end{aligned} \tag{12}$$

As a result, we can confirm that the order of the embedded formula is 3, as shown in Figure 3. For comparison, the original 6-th order y_k is also plotted in the same figure.

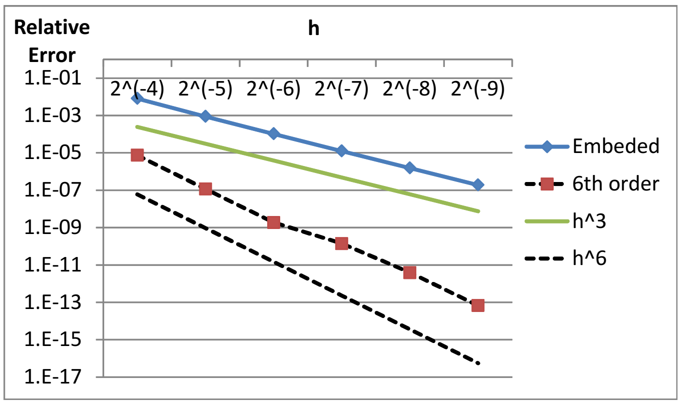

Figure 3: Relative error of the 3rd order embedded formula derived from 3-stage 6th order IRK formula

In our implementation of the MP ODE solver based on high-order IRK Gauss formulas, we currently use the following value $\|\mathbf{err}_k\|$ for local error estimation[5].

$$\|\mathbf{err}_k\| = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(\frac{\left|\hat{y}_j^{(k+1)} - y_j^{(k+1)}\right|}{ATOL + RTOL \cdot \max\left(\left|y_j^{(k+1)}\right|, \left|y_j^{(k)}\right|\right)}\right)^2}$$

The next step size h_{k+1} at x_{k+1} is set as

$$h_{k+1} := 0.9\|\mathbf{err}_k\|^{1/(m+1)} h_k.$$

4.3 A-stability Regions of Embedded Formulas

One advantage of IRK formulas can be A-stable; however, the embedded formulas derived from the original IRK formulas are not A-stable. Actually, the stability region of Hairer's embedded formula is narrower than that of the original Radau IIA formula, as shown in Figure 4.

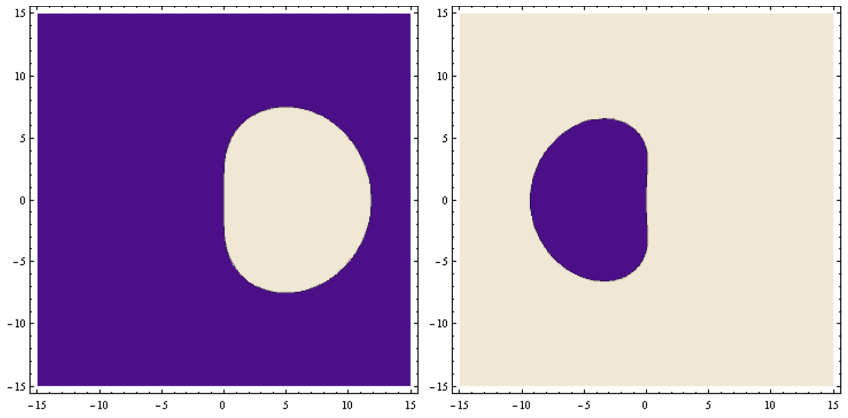

Figure 4: Stability regions: 3-stage 5th order Radau IIA formula (left) and the corresponding Hairer's embedded formula (right).

We must select the parameter $\gamma_0 \neq 0$ at which the embedded formula can have a wider region. As a result, we currently consider that $\gamma_0 = 1/8$ is better because it is not too small and it can be expressed in powers of 2. In addition, its region is wider than that of Hairer's embedded formula. The stability region is shown in Figure 5.

5 Numerical Experiments

As described in previous sections, the following three techiques are applied to our MP ODE solver based on high-order IRK formulas:

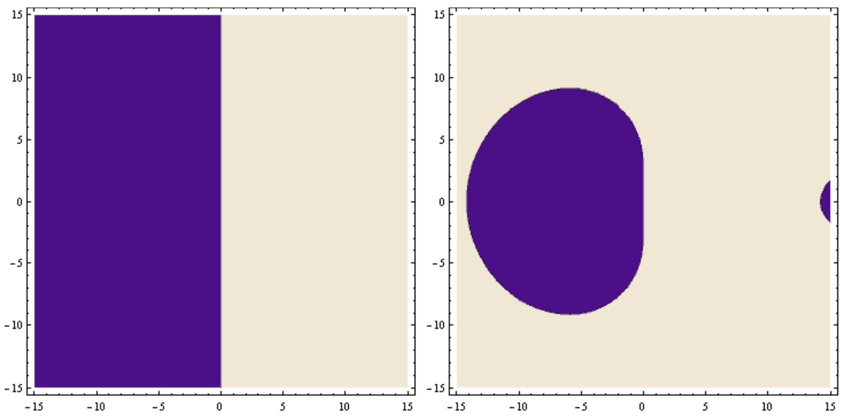

Figure 5: Stability regions: 3-stage 6th order Gauss formula (left) and our embedded formula(left, $\gamma_0 = 1/8$)

1. Simplified Newton Method with SPARK3 type reduction in inner iteration
2. DP-MP type mixed precision iterative refinement method to accelarate inner iteration
3. Step size selection based on embedded formulas automatically derived

In addition, users can select DP and MP Krylov subspace methods supporting a banded Jacobi matrix in the same way as SPARK3.

To evaluate the peformance of our implementation, we present the results of numerical experiments. All computations are executed on a Intel Core i7 920 + CentOS 5.4 x86_64 machine (gcc 4.1.2 + BNCpack 0.8 + MPFR 3.1.0/GMP 5.0.2).

5.1 Non Stiff Problem

The Lorenz problem (13) is a well-known problem in complex systems, and it is not stiff; however, the accuracy of approximation is worse in a longer integration interval. Thus, we must use MP arithmetic in propotion to the length of the integration interval.

$$\left\{ \begin{array}{rcl} \dfrac{d\mathbf{y}}{dx} & = & \begin{bmatrix} \sigma(-y_1 + y_2) \\ -y_1 y_3 + r y_1 - y_2 \\ y_1 y_2 - b y_3 \end{bmatrix} \\ \mathbf{y}(0) & = & [0\ 1\ 0]^T \end{array} \right. , \qquad (13)$$

Integration Interval:$[0, 50]$

Table 2: Lorenz Problem

	$RTOL = 10^{-30}, ATOL = 0$		$RTOL = 10^{-50}, ATOL = 0$	
	10 stages	15 stages	10 stages	15 stages
# steps	41137	5112	2709021	91169
Comp.Time (s)	192.0	64.3	12911.1	1112.2
Max.Rel.Error	3.9×10^{-19}	4.4×10^{-19}	3.8×10^{-39}	4.9×10^{-39}
Min.Rel.Error	7.3×10^{-21}	8.3×10^{-21}	7.1×10^{-41}	9.1×10^{-41}

where $\sigma = 10$, $r = 470/19$, and $b = 8/3$. In the case of the above integration interval, we lose around 13 decimal digits of the approximation of $\mathbf{y}(50)$. Hence, we select 70 decimal digits (233 bits), 10-stage 20th order and 15-stage 30th order Gauss formulas. The numerical results are shown in Table 2.

The numerical results using both 10-stage and 15-stage formulas indicate that we can obtain the appropriate accuracy of corresponding $RTOL$s.

5.2 Stiff Problem

Next, we solve the van del Pol equation (14), which is provided in Testset[4].

$$\begin{cases} \dfrac{d\mathbf{y}}{dx} = \begin{bmatrix} y_2 \\ ((1-y_1^2)y_2 - y_1)/10^{-6} \end{bmatrix} \\ \mathbf{y}(0) = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \end{cases} \tag{14}$$

Integration Interval:$[0, 2]$

By using a 15-stage 30th order Gauss formula computed in 50 decimal digits (167 bits) MP arithmetic, we set $ATOL = 0$ and $RTOL = 10^{-40}$ or 10^{-30}. In this case, Figure 6 shows the history of $||\mathbf{err}_k||$ and step size.

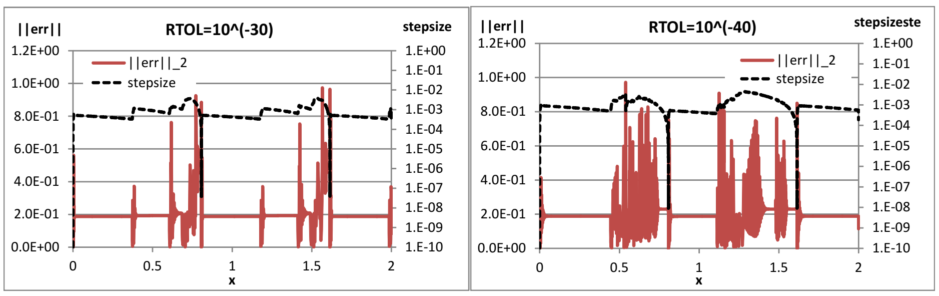

Figure 6: History of local error estimation and step size: van der Pol equaiton

Table 3: Van del Pol equation: The number of steps, computational time and relative errors.

	$RTOL = 10^{-30}$	$RTOL = 10^{-40}$
# steps	4325	6202
Comp.Time (s)	127.8	208.9
Max.Rel.Error Min.Rel.Error	1.2×10^{-29} 2.2×10^{-36}	1.0×10^{-39} 8.6×10^{-47}

As a result, we can obtain the appropriate approximations corresponding to each $RTOL$s listed in Table 3.

5.3 1-Dimensional Brusselator Problem

As a large-scale problem, we solve the 1-dimensional Brusselator problem[8] by using our MP ODE solver.

$$\begin{cases} \frac{\partial u}{\partial t} = 1 + u^2 v - 4 + 0.02 \cdot \frac{\partial^2 u}{\partial x^2} \\ \frac{\partial v}{\partial t} = 3u - u^2 v + 0.02 \cdot \frac{\partial^2 v}{\partial x^2} \end{cases} \tag{15}$$

The above original partial differential equation (15) can be discretetized as a large-scale ODE as follows:

$$\begin{cases} \frac{du_i}{dt} = 1 + u_i^2 v_i - 4 + 0.02 \cdot \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} \\ \frac{dv_i}{dt} = 3u_i - u_i^2 v_i + 0.02 \cdot \frac{v_{i+1} - 2v_i + v_{i-1}}{(\Delta x)^2} \\ u_0(t) = u_{N+1}(t) = 1, \ v_0(t) = v_{N+1}(t) = 3, \\ u_i(0) = 1 + \sin(2\pi i \Delta x), \ v_i(0) = 3 \end{cases} \quad (i = 1, 2, ..., N) \tag{16}$$

Integration Interval:$[0, 10]$

We solve the above ODE for the following situation:

Parameters $N = 500, n = 2N = 1000, \Delta x = 1/(N+1) = 1/501$

TOLs $RTOL = ATOL = 10^{-30}$

Used linear solver DP-MP($L = 50$) type mixed precision BiCGSTAB method with band matrix-vector multiplication.

In many large-scale problems, J can be sometimes expressed as a sparse matrix. In particular, for an MP enviroment, we must treat J as a sparse matrix in order to overcome the limitation of main memory. In this case, the dense matrix of J need about 36 MB for 50 decimal digits, and hence $3 \times 36 \times 10 = 1.08$ GB

Table 4: 1-dimensional Brusselator Problem

	Left preconditioned DP BiCGSTAB		Unpreconditioned	
	10 stages	20 stages	10 stages	20 stages
# steps	2966	341	15056	7587
Comp.Time (s)	11205	3642	77717	64573
Max.Rel.Error	2.2×10^{-25}	1.3×10^{-20}	3.9×10^{-25}	3.8×10^{-25}
Min.Rel.Error	1.3×10^{-29}	6.1×10^{-23}	2.1×10^{-27}	2.1×10^{-27}

for a 10-stage IRK method. On the other hand, the band matrix of J need be about 0.18 MB, and hence, $3 \times 0.18 \times 10 = 5.4$ MB. This problem is sutable for DP-MP Krylov subspace methods in inner iteration. In this case, we use the left preconditioned DP-MP BiCGSTAB and normal DP-MP BiCGSTAB methods.

Table 4 shows the result of numerical experiments for (16). The left preconditioned DP BiCGSTAB methods at (4) and (6) in the DP-MP type iterative refinement method is needed in order to be converged successfully.

By comparing the numerical values provided in SPARK3, we confirmed that all elements of the approximations have over true 14 decimal digits. Moreover, we can find that they have about 20 - 29 true decimal digits by comparing the results obtained by using 100 decimal digits MP computation. In the above-mentioned problems, faster computations with the same accuracy of approximations can be achieved with higher-order formulas.

However, there are some problems to be solved in order to speed up our MP ODE solver as shown Figure 7.

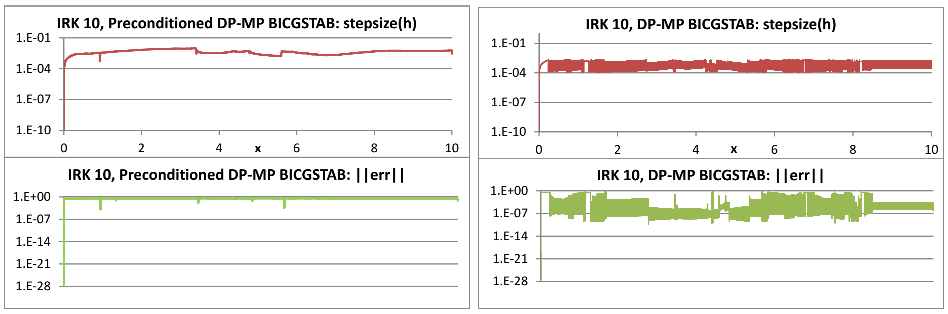

Figure 7: History of step size and $||\mathbf{err}_k||$: 10 stages, 20th order : Left preconditioned DP-MP BiCGSTAB(Left), DP-MP BiCGSTAB(Right)

The two sets of graphs of $||\mathbf{err}_k||$ and step size show that the right ones are widely wiggled because of non-convergence of the unpreconditioned DP BiCGSTAB method used in the iterative refinement method. If the DP BiCGSTAB method is not convergent, the approximation computation is rejected and a new one is re-computed after the step size becomes smaller at the discretized point. These bottlenecks can be overcome by using predconditionings or other robust DP iterative methods for a system of linear equations in inner iteration.

6 Conlusion and Future Works

We showed that our MP ODE solver based on high-order IRK formulas can obtain accurate approximations in some problems. However, some problems remain such as the unsupported DP robust linear solver for sparse Jacobi matrix.

Our final objective is to provide practical and high-performance DP and MP ODE solvers based on high-order IRK methods.

In order to achieve this objective, we plan to tackle the following issues in the future:

1. Parallelization of inner iteration in IRK method for multi-core CPU and GPU. For this purpose, we plan to use a well-tuned linear computation library based on LAPACK and BLAS.

2. We plan to accumulate many numerical experiments, especially for ill-conditioned problems requiring MP arithmetic, and we also plan to provide some selection for solvers of system of linear equations in inner iteration in order to optimize the computational time and user-required accuracy.

Acknowledgments

I thank Hideko Nagasaka and Masatsugu Tanaka for encouraging me when writing my doctoral thesis that is one of origins of this paper.

References

[1] The GNU Multiple Precision arithmetic library, http://gmplib.org/.

[2] A.Buttari, J.Dogarra, Julie Langou, Julien Langou, P.Luszczek, and J.Karzak, Mixed precision iterative refinement techniques for the solution of dense linear system. *The International Journal of High Performance Computing Applications*, Vol. 21, No. 4, pp. 457–466, 2007.

[3] S.P.Nørsett E.Hairer and G.Wanner, *Solving Ordinary Differential Equations*. Springer-Verlarg, 1996.

[4] F. Mazzia, C. Magherini and J. Kierzenka, Test set for IVP solvers. http://www.dm.uniba.it/~testset/testsetivpsolvers/.

[5] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II*, Springer-Verlarg, 1996.

[6] L.O. Jay and T. Braconnier, A parallelizable preconditioner for the iterative solution of implicit Runge-Kutta-type methods. *Journal of Computational and Applied Mathematics*, Vol. 111, pp. 63–76, 1999.

[7] Julie Langou, Julien Langou, Piotr Luszczek, Jakub Kurzak, Alfredo Buttari, and Jack J. Dongarra, Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems). Technical Report 175, LAPACK Working Note, June 2006.

[8] L.O.Jay, SPARK3. http://www.math.uiowa.edu/~ljay/SPARK3.html.

[9] C. B. Moler, Iterative refinement in floating point. *Journal of the ACM*, pp. 316 – 321, 1967.

[10] MPFR Project, The MPFR library. http://www.mpfr.org/.