# 混合精度反復改良法を用いた多倍長陰的 Runge-Kutta 法の高速化

静岡理工科大学
幸谷智紀
http://na-inet.jp/

第 3 回多倍長精度計算フォーラム
2013 年 3 月 8 日 (金)

# 概要

# 研究の経緯

BNCpack(on MPFR/GMP) を作り続けて 10 数年 ··· 。
2009 年から本格的に混合精度反復改良法を陰的
Runge-Kutta(IRK) 法に取り込む。

1. 倍精度・多倍長精度の陽的 Runge-Kutta 法, 補外法 (Gragg 法), 陰的 Runge-Kutta(IRK) 法を常微分方程式 (ODE) の初期値問題を素朴に実装 (2001 年 — 2008 年)
2. IRK 法に Buttari らによる直接法ベースの混合精度反復改良法の取り込み→線型 ODE に対して高速化 (cf. 応用数理学会論文誌, Vol.19, No.3, pp.313-328, 2009)
3. 複素対角化法の実装→困難性が明らかに (2011 年)
4. L.O.Jay によるブロック三重対角化法の実装 (2012 年 3 月)
5. 埋め込み型公式採用の刻み幅制御完成→非線型 ODE 対応 (2012 年 9 月)

# 混合精度反復改良法とは？

$S$ 桁計算と $L$ 桁計算 ($S << L$) を組み合わせて計算時間を削減する手法。

線型方程式: $\boxed{C\mathbf{x} = \mathbf{d}}$, $C \in \mathbb{R}^{N\times N}$, $\mathbf{d}$, $\mathbf{x} \in \mathbb{R}^N$

$\Longrightarrow$

$(L)$ Solve $C\mathbf{x}_0 = \mathbf{d}$ for $\mathbf{x}_0$.

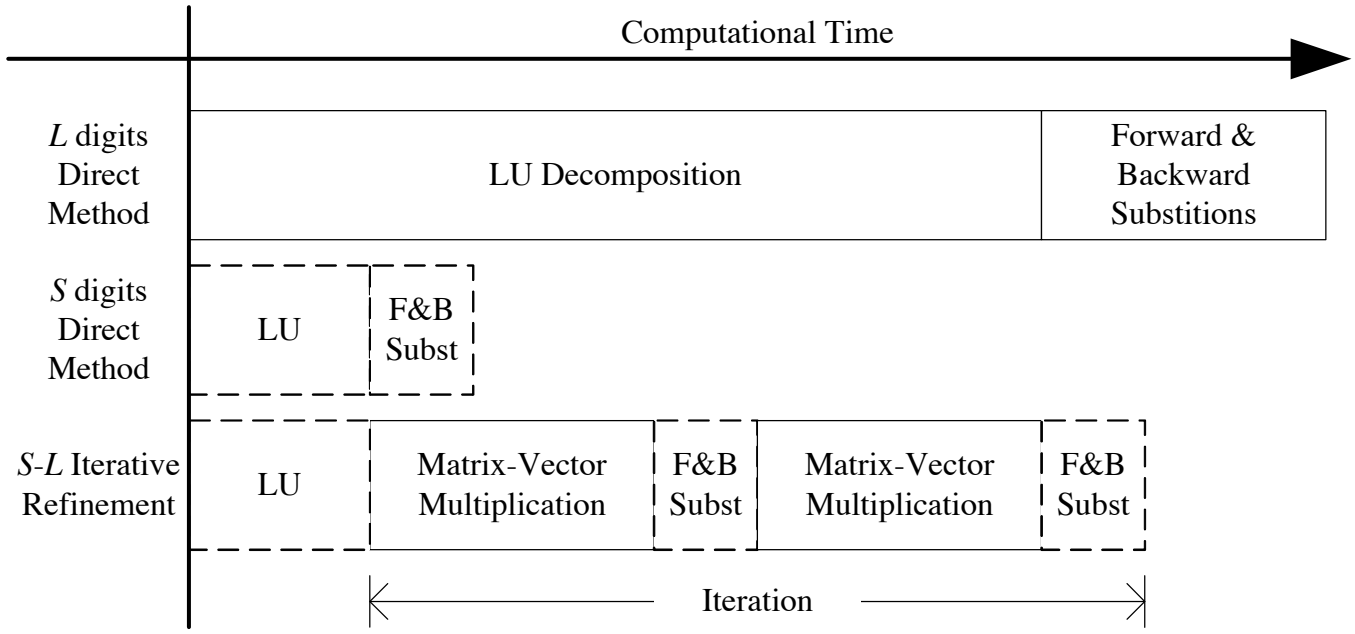
For $\nu = 0, 1, 2, ...$

- $(L)$ $\mathbf{r}_\nu := \mathbf{d} - C\mathbf{x}_\nu$
- $(S)$ $\mathbf{r}'_\nu := \mathbf{r}_\nu / \|\mathbf{r}_\nu\|$
- $(S)$ Solve $C\mathbf{z} = \mathbf{r}'_\nu$ for $\mathbf{z}$.
- $(L)$ $\mathbf{x}_{\nu+1} := \mathbf{x}_\nu + \|\mathbf{r}_\nu\|\mathbf{z}$
- Check convergence.

$\Longrightarrow \mathbf{x} := \mathbf{x}_{\nu_{stop}}$

(cf.) Buttari, Alfredo, et al. International Journal of High Performance Computing Applications 21.4 (2007): 457-466.

# 理想的なケース … 直接法の場合




但し，次のような特性 (欠点) がある。

- $S$ 桁計算可能な良条件問題に限定。
- $L$ が大きくなると残差計算が重くなる。
- $L >> S$ となると，反復回数が増える。

結論 1：使えるケースは限られる (GPU 上での単精度・倍精度組み合わせ等)

結論 2: IRK 法は数少ない使えるケース！(と思っている)

# 常微分方程式の初期値問題と Runge-Kutta 法 (1/2)

常微分方程式の初期値問題:

$$\begin{cases} \dfrac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y}) \in \mathbb{R}^n \\ \mathbf{y}(t_0) = \mathbf{y}_0 \in \mathbb{R}^n \end{cases}$$

積分区間: $[t_0, \alpha]$

$m$ 段 Runge-Kutta 法:

離散化: $t_0,\ t_1 := t_0 + h_0,\ \ldots,\ \underline{t_{k+1} := t_k + h_k},\ \ldots$

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

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

$$\approx \mathbf{y}(t_{k+1})$$

# 陰的 Runge-Kutta 法

Runge-Kutta 法の係数:

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

陰的 Runge-Kutta(IRK) 法の係数の例：3 段 6 次 Gauss 型

$$
\begin{array}{c|ccc}
\frac{5-\sqrt{15}}{10} & \frac{5}{36} & \frac{10-3\sqrt{15}}{45} & \frac{25-6\sqrt{15}}{180} \\
\frac{1}{2} & \frac{10+3\sqrt{15}}{72} & \frac{2}{9} & \frac{10-3\sqrt{15}}{72} \\
\frac{5+\sqrt{15}}{10} & \frac{25+6\sqrt{15}}{180} & \frac{10+3\sqrt{15}}{45} & \frac{5}{36} \\
\hline
 & \frac{5}{18} & \frac{8}{18} & \frac{5}{18}
\end{array}
$$

# $m$ 段 $2m$ 次 Gauss 型 IRK 公式

$c_p$ :$m$ 次シフト Legendre 直交多項式 $\tilde{P}_m(x)$ の零点

$$a_{pq} = \int_0^{c_p} \prod_{j=1, j\neq q}^{m} \frac{x - c_j}{c_q - c_j} dx \ (p, q = 1, 2, ..., m)$$

$$b_p = \int_0^{1} \prod_{j=1, j\neq q}^{m} \frac{x - c_j}{c_q - c_j} dx \ (p = 1, 2, ..., m)$$

長所
- $m$ 段の時 $2m$ 次 (最高次数)
- A 安定, B 安定，かつ，Simplectic 解法

短所
- 次のステップの計算のために毎回非線型方程式 $(*)$ を解く必要がある
  $\Rightarrow$ここを高速化しないと使い物にならない解法

# 準 Newton 法 (1/3)

Initial guess: $\mathbf{k}_1^{(0)}$, ..., $\mathbf{k}_m^{(0)}$

$$\begin{bmatrix} \mathbf{k}_1^{(l+1)} \\ \mathbf{k}_2^{(l+1)} \\ \vdots \\ \mathbf{k}_m^{(l+1)} \end{bmatrix} := \begin{bmatrix} \mathbf{k}_1^{(l)} \\ \mathbf{k}_2^{(l)} \\ \vdots \\ \mathbf{k}_m^{(l)} \end{bmatrix}$$

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

# 準 Newton 法 (2/2)

ここで，$J(\mathbf{k}_1^{(l)}, \mathbf{k}_2^{(l)}, ..., \mathbf{k}_m^{(l)}) \in \mathbb{R}^{mn \times mn}$ は

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

但し，

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

$I_n : n$ 次元単位行列

である。

Newton 法の反復計算をサボるため，$J$ を下記のように固定。

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

# 線型 ODE の性能評価 1

## テスト問題

一様乱数を用いて生成した正則行列 $R$ と逆行列 $R^{-1}$ を用いて

$$\begin{cases} \dfrac{d\mathbf{y}}{dt} = -(R\ \mathrm{diag}(n, n-1, ..., 1)\ R^{-1})\ \mathbf{y} \\ \\ \mathbf{y}(0) = [1\ ...\ 1]^T \end{cases}$$

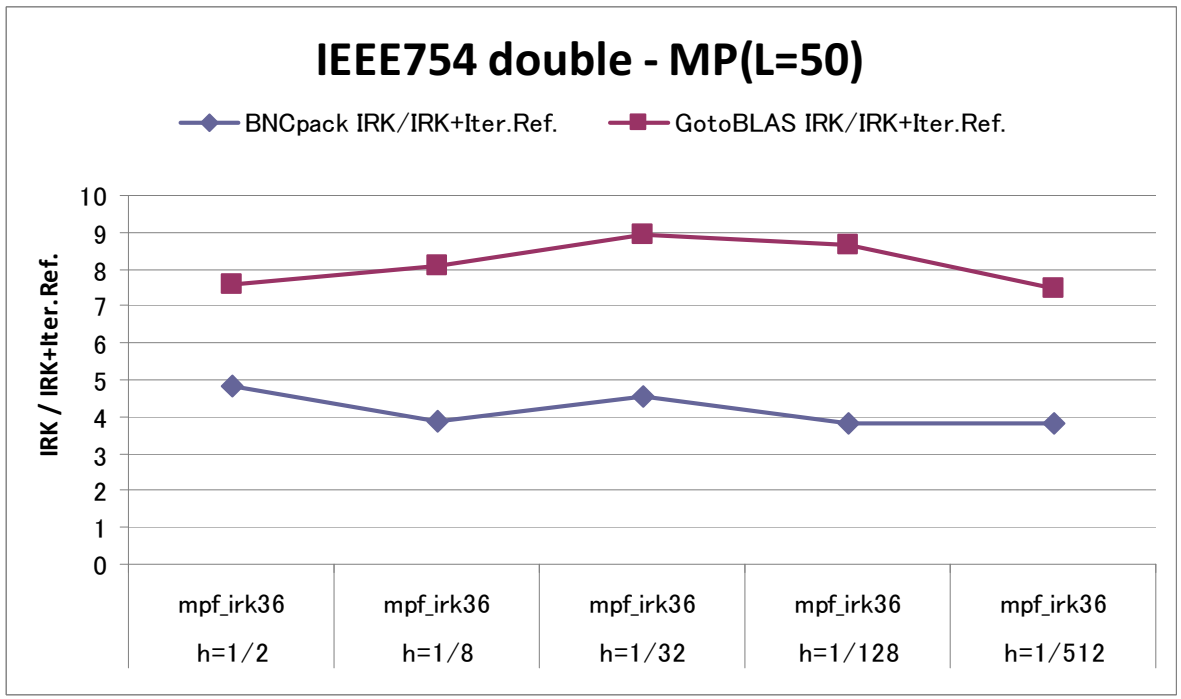
積分区間:$[0, 20]$

という定係数常微分方程式を作成。$n = 128$, $S = 15$(倍精度 DP), $L = 50$ 桁計算 (MP) した時の性能評価を行う.

# 線型 ODE の性能評価 1

## 準 Newton 法を適用した場合 (2009 年の結果)

$h = 1/512$ →約 24 桁の近似解が得られる直接法ベースの DP-MP 型反復改良法を使用した 3 段 6 次 IRK 法の性能向上比 (対 MP 直接法)




直接法を用いた時に比べ，GotoBLAS を使用した場合で約 7.5〜8.9 倍 BNCpack を使用した場合でも約 3.8〜4.8 倍の性能向上が得られた.

# 簡易 Newton 法における係数行列リダクション

## 簡易 Newton 法

準 Newton 法の係数行列 $J(\mathbf{k}_1^{(0)}, \mathbf{k}_2^{(0)}, ..., \mathbf{k}_m^{(0)})$ を更に簡易化

$$J_{pq} := h_k a_{pq} \frac{\partial}{\partial \mathbf{y}} \mathbf{f}(t_k, \mathbf{y}_k) = h_k a_{pq} J$$

更に，$\mathbf{k}_i^{(l)}$ の代わりに $Y_i$ を使って
$Y_i = \mathbf{y}_k + h_k \sum_{j=1}^{m} a_{ij} \mathbf{f}(t_k + c_i h_k, Y_j)$ と書くと，解くべき連立一次方程式は次のように表現できる。

$$(I_m \otimes I_n - h_k A \otimes J)\mathbf{Z} = -\mathbf{F}(\mathbf{Y}) \tag{1}$$

ここで

$$\mathbf{F}(\mathbf{Y}) = [\cdots Y_i - \mathbf{y}_k - h_k \sum_{j=1}^{m} a_{ij} \mathbf{f}(t_k + c_i h_k, Y_j) \cdots]^T$$

# RADAU5 型リダクションと SPARK3 型リダクション

既存の倍精度 IRK ライブラリ

RADAU5 by Hairer,
`http://www.unige.ch/~hairer/prog/stiff/radau5.f`

SPARK3 by Jay,
`http://www.math.uiowa.edu/~ljay/spark3.tar.gz`

は簡易 Newton 法を使用し，(1) における係数行列 $I_m \otimes I_n - h_k A \otimes J$ をリダクションする。

RADAU5 型 $A$ を (複素) 対角化→相似変換行列が悪条件＆複素演算が必須

SPARK3 型 $A$ を実三重対角化→相似変換行列が良条件＆実数演算のみ

$\Longrightarrow$ SPARK3 型リダクションを採用

# SPARK3 型リダクション (1/3)

W 変換 (Hairer & Wanner) を用いた実三重対角化 (Gauss 型の場合)

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

ここで

$$w_{ij} = \tilde{P}_{j-1}(c_i) \ (i, j = 1, 2, ..., m)$$

$$\zeta_i = \left(2\sqrt{4i^2 - 1}\right)^{-1} \quad (i = 1, 2, ..., m-1)$$

$$B = \mathsf{diag}(\mathbf{b}), \ I_m = W^T BW = \mathsf{diag}(1\ 1\ \cdots\ 1)$$

# SPARK3 型リダクション (2/3)

よって，簡易 Newton 法の連立一次方程式の係数行列は

$$(W^T B \otimes I_n)(I_m \otimes I_n - h_k A \otimes J)(W \otimes I_n)$$

$$= I_m \otimes I_n - h_k X \otimes J = \begin{bmatrix} E_1 & F_1 & & & \\ G_1 & E_2 & F_2 & & \\ & \ddots & \ddots & \ddots & \\ & & G_{m-2} & E_{m-1} & F_{m-1} \\ & & & G_{m-1} & E_m \end{bmatrix}$$

となる。ここで

$$E_1 = I_n - \frac{1}{2} h_k J,\ E_2 = \cdots = E_s = I_n$$

$$F_i = h_k \zeta_i J,\ G_i = -h_k \zeta_i J\ (i = 1, 2, ..., m-1)$$

となる。

# SPARK3 型リダクション (3/3)

更に左前処理行列 $P$ として

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

を作り,

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

を $\mathbf{Z}$ について解く。

# SPARK3 型リダクション + 簡易 Newton 法 + 混合精度反復改良法のアルゴリズム

1. $\mathbf{Y}_{-1} := [\mathbf{y}_k\ \mathbf{y}_k\ \ldots\ \mathbf{y}_k]^T \in \mathbb{R}^{mn}$
2. For $l = 0, 1, 2, \ldots$ ··· 簡易 Newton 法

   $\mathbf{Y}_l := [Y_1^{(l)}\ Y_2^{(l)}\ \ldots\ Y_m^{(l)}]^T$ ここで $Y_i^{(l)} = \mathbf{y}_0 + h_k \sum_{j=1}^{m} a_{ij}\mathbf{f}(t_k + c_i h_k, Y_i^{(l-1)})$

   $C := I_m \otimes I_n - h_k X \otimes J$, $\mathbf{d} := (W^T B \otimes I_n)(-\mathbf{F}(\mathbf{Y}_l))$

   $(S)$ Solve $C\mathbf{x}_0 = \mathbf{d}$ for $\mathbf{x}_0$

   For $\nu = 0, 1, 2, \ldots$ ··· 混合精度反復改良法

   $\mathbf{r}_\nu := \mathbf{d} - C\mathbf{x}_\nu$

   $(S)$ $\mathbf{r}'_\nu := \mathbf{r}_\nu / \|\mathbf{r}_\nu\|$

   $(S)$ Solve $C\mathbf{z} = \mathbf{r}'_\nu$ for $\mathbf{z}$

   $\mathbf{x}_{\nu+1} := \mathbf{x}_\nu + \|\mathbf{r}_\nu\|\mathbf{z}$

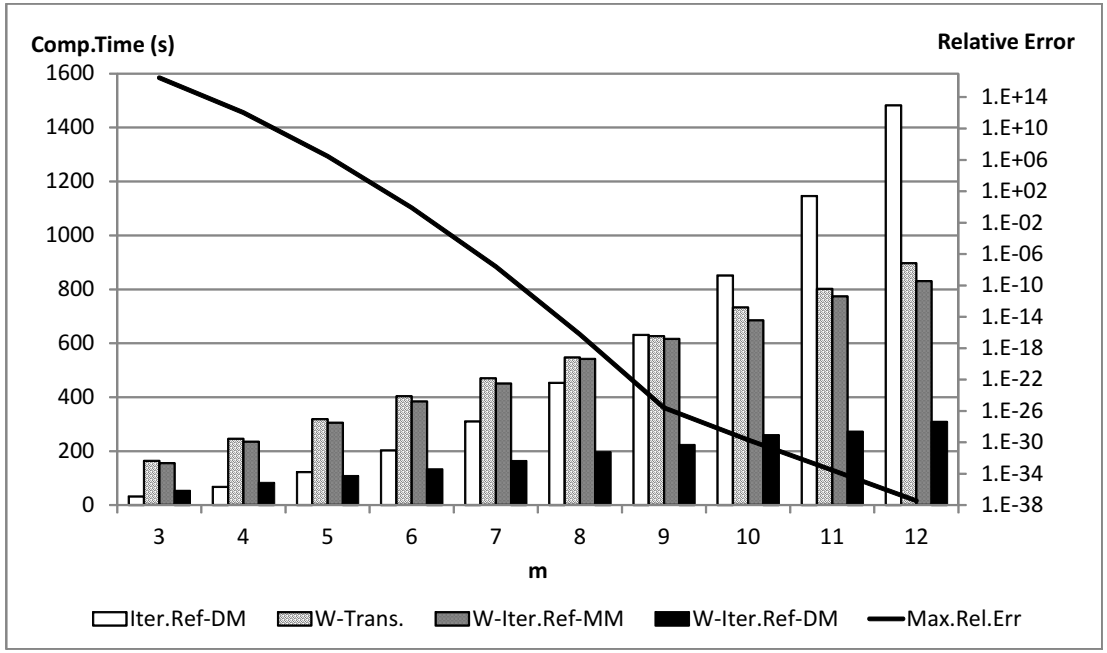
   Check convergence $\Rightarrow \mathbf{x}_{\nu_{stop}}$

   $\mathbf{Y}_{l+1} := \mathbf{Y}_l + (W \otimes I_n)\mathbf{x}_{\nu_{stop}}$

   Check convergence $\Rightarrow \mathbf{Y}_{l_{stop}}$
3. $\mathbf{Y} := \mathbf{Y}_{l_{stop}} = [Y_1\ Y_2\ \ldots\ Y_m]^T$
4. $\mathbf{y}_{k+1} := \mathbf{y}_k + h_k \sum_{j=1}^{m} b_j \mathbf{f}(t_k + c_j h_k, Y_j)$

# 線型 ODE の性能評価 2

計算環境 ⋯ Intel Core i7 920, 8GB RAM, CentOS 5.6 x86_64, gcc 4.1.2, MPFR 3.1.1/GMP 5.0.5




4 方法とも相対誤差は全く同じ

Iter.Ref-DM リダクションなし準 Newton 法+倍精度 (DP)-多倍長 (MP) 混合精度反復改良法（直接法ベース）

W-Trans. SPARK3 リダクション+ MP 直接法

W-Iter.Ref-MM SPARK3 リダクション + MP($S = L/2$)-MP 混合精度反復改良法

W-Iter.Ref-DM SPARK3 リダクション + DP-MP 混合精度反復改良法

# 埋め込み型公式を用いた刻み幅制御

$\gamma_0 = 0.125$ として与え，以下の線型方程式を解く。

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

$\Longrightarrow$ $m$ 次の埋込型公式に基づく近似値 $\hat{\mathbf{y}}_{k+1}$:

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

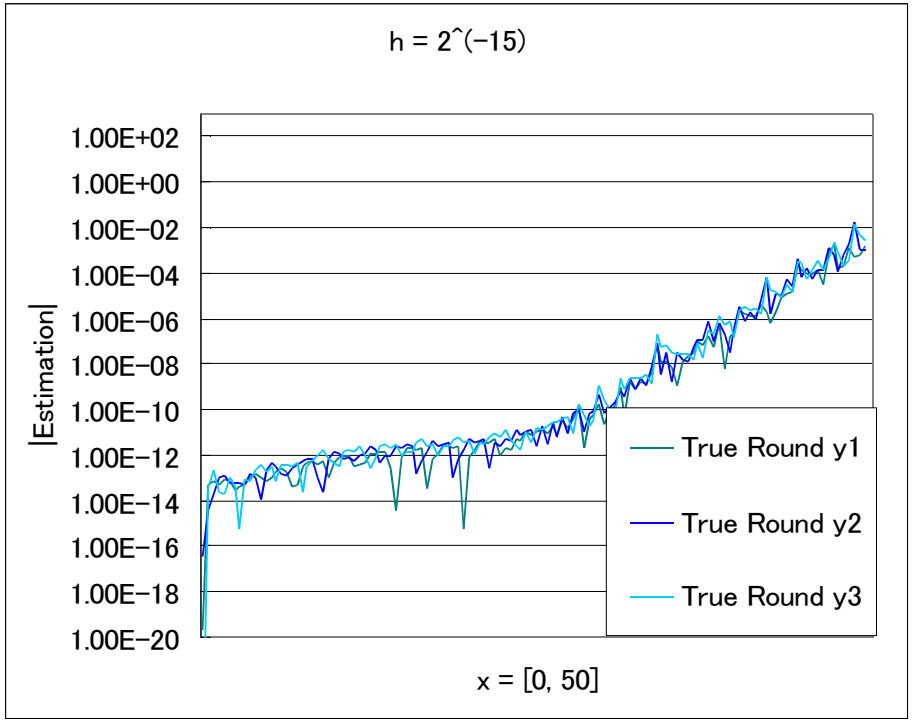
下記を満足するまで $h_k$ を小さくする。

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

# 非線型 ODE の性能評価 1

$$\begin{cases} \dfrac{d\mathbf{y}}{dt} = \begin{bmatrix} \sigma(-y_1 + y_2) \\ -y_1 y_3 + r y_1 - y_2 \\ y_1 y_2 - b y_3 \end{bmatrix}, \sigma = 10, r = 470/19, b = 8/3 \\ \mathbf{y}(0) = [0\ 1\ 0]^T \end{cases}$$

積分区間:$[0, 50]$




約 15 桁精度が落ちる。

# Lorenz 問題 (Non-stiff) の計算結果

直接法ベースの DP-MP 混合精度反復改良法使用。

| 150 dec.digits | $RTOL = 10^{-100}, ATOL = 0$ | | |
|---|---|---|---|
| # stages($m$) | 30 | 50 | 100 |
| Comp.Time(s) | 12273.8 | 2348.9 | 2215.4 |
| # steps | 106095 | 5293 | 573 |
| Ave.Comp.Time(s) | 0.1 | 0.4 | 3.9 |
| Max.Rel.Error | 9.7E-92 | 1.5E-89 | 1.5E-89 |
| Min.Rel.Error | 1.8E-93 | 2.9E-91 | 2.8E-91 |
| | $RTOL = 10^{-120}, ATOL = 0$ | | |
| # stages($m$) | 30 | 50 | 100 |
| Comp.Time(s) | 53318.6 | 6681.9 | 3616.2 |
| # steps | 468694 | 13057 | 884 |
| Ave.Comp.Time(s) | 0.1 | 0.5 | 4.1 |
| Max.Rel.Error | 4.4E-110 | 8.5E-110 | 2.7E-109 |
| Min.Rel.Error | 8.3E-112 | 1.6E-111 | 5.1E-111 |

- 刻み数 (# steps) 削減効果が高次公式利用時の低速さを上回ると高速
- $RTOL = 10^{-120}$ の場合は更に高次の方が高速に

# 非線型 ODE の性能評価 2

## Brusselator 1 次元問題 (Stiff)

下記の時間発展偏微分方程式を

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

次のように ODE 化 (1000 次元).

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

$(i = 1, 2, ..., N = 500)$

積分区間 $[0, 10]$

# Brusselator 問題の計算結果

帯行列対応 + 左前処理つき BiCGSTAB 法 (倍精度) を使用。

| 70 dec.digits | $RTOL = ATOL = 10^{-40}$ | | |
|---|---|---|---|
| # stages($m$) | 15 | 20 | 30 |
| Comp.Time(s) | 22097.0 | 23573.3 | 23776.9 |
| # steps | 2173 | 1424 | 805 |
| Ave.Comp.Time(s) | 10.2 | 16.6 | 29.5 |
| Max.Rel.Error | 3.3E-34 | 3.2E-30 | 1.0E-29 |
| Min.Rel.Error | 8.8E-37 | 1.5E-32 | 4.9E-32 |
| | $RTOL = ATOL = 10^{-50}$ | | |
| # stages($m$) | 15 | 20 | 30 |
| Comp.Time(s) | 72285.1 | 30996.0 | 30494.4 |
| # steps | 6098 | 1628 | 860 |
| Ave.Comp.Time(s) | 11.9 | 19.0 | 35.5 |
| Max.Rel.Error | 6.3E-45 | 2.9E-41 | 2.1E-38 |
| Min.Rel.Error | 2.8E-47 | 1.4E-43 | 1.0E-40 |

収束すれば，Krylov 部分空間法の方が直接法より高速になる傾向がある。

# 今後の課題

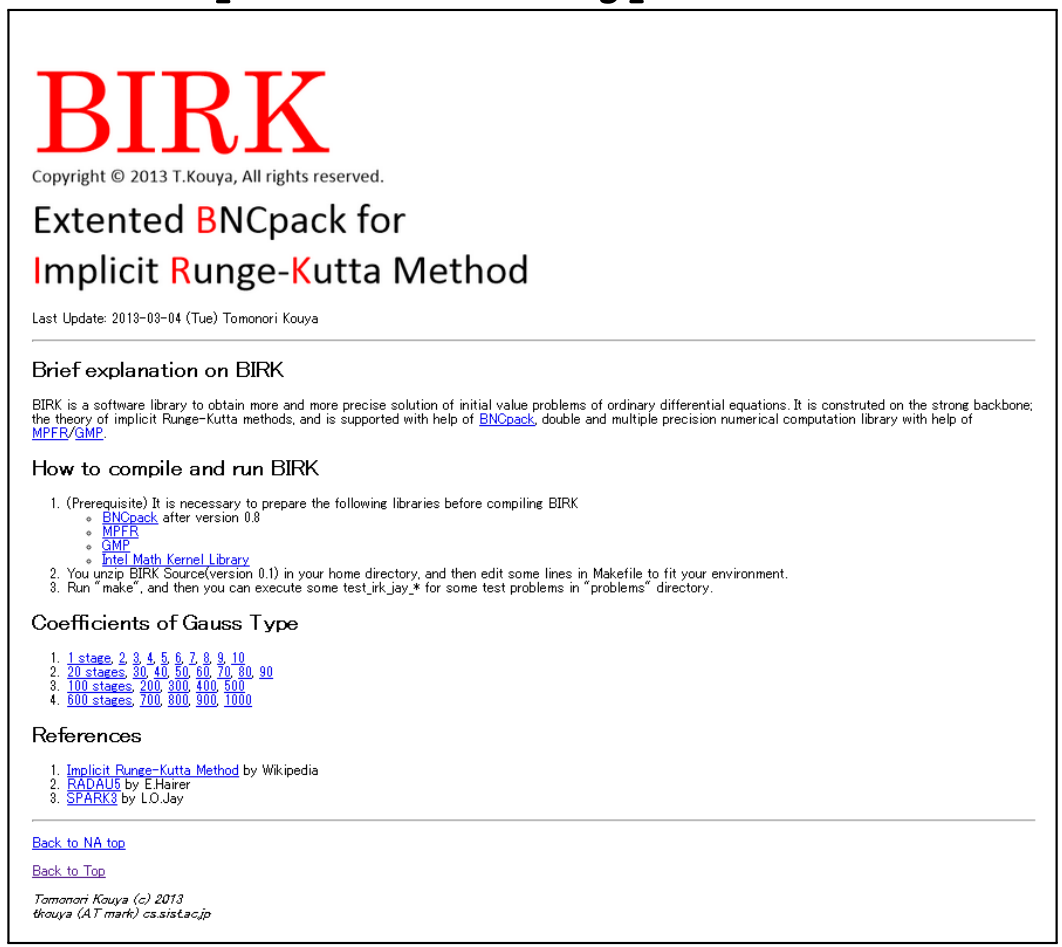
フレームワークは出来たが，詳細な検証はまだこれから。

- アルゴリズム全体に関してはまだ工夫の余地あり。
  - 倍精度計算で安定的に収束する反復解法の探求
  - $J$ の評価回数の削減
  - 刻み幅不採用時の高速化
    ⋯ 等々
- 高性能化を目指す。

  倍精度 IRK → GPGPU への適用 (既存ライブラリの応用)
  多倍長 IRK →並列化！(一部対応)
- 大規模な時間発展偏微分方程式に適用
  (Kuramoto-Sivashinsky 方程式等)

# BIRK — extended Bncpack for Implicit Runge-Kutta methods

http://na-inet.jp/na/birk/



BIRK



Extented BNCpack for
Implicit Runge-Kutta Method

Last Update: 2013-03-04 (Tue) Tomonori Kouya

Brief explanation on BIRK

BIRK is a software library to obtain more and more precise solution of initial value problems of ordinary differential equations. It is construted on the strong backbone; the theory of implicit Runge-Kutta methods, and is supported with help of BNCpack, double and multiple precision numerical computation library with help of MPFR/GMP.

How to compile and run BIRK

1. (Prerequisite) It is necessary to prepare the following libraries before compiling BIRK
   - BNCpack after version 0.8
   - MPFR
   - GMP
   - Intel Math Kernel Library
2. You unzip BIRK Source(version 0.1) in your home directory, and then edit some lines in Makefile to fit your environment.
3. Run "make", and then you can execute some test_irk_jay_* for some test problems in "problems" directory.

Coefficients of Gauss Type

1. 1 stage, 2, 3, 4, 5, 6, 7, 8, 9, 10
2. 20 stages, 30, 40, 50, 60, 70, 80, 90
3. 100 stages, 200, 300, 400, 500
4. 600 stages, 700, 800, 900, 1000

References

1. Implicit Runge-Kutta Method by Wikipedia
2. RADAU5 by E.Hairer
3. SPARK3 by L.O.Jay

Back to NA top

Back to Top

*Tomonori Kouya (c) 2013*
*tkouya (AT mark) cs.sist.ac.jp*

IRK 公式の係数，ソースコードを公開。