並列多倍長陰的 Runge-Kutta 法を用いた 蔵本-Sivashinsky 方程式の高精度計算

静岡理工科大学 幸谷智紀

HPCS2014 2014 年 1 月 7 日 (火) — 8 日 (水)

1 目的

複雑系常微分方程式 (ODE)，偏微分方程式 (PDE) の高精度な解を求めるためには高次の離散化を行い，多倍長計算を併用する必要がある。我々は任意次数の Gauss 型陰的 Runge-Kutta(IRK) 法に基づく並列多倍長 ODE ソルバー BIRK(extented Bncpack for IRK methods) を実装した [2]。ここでは 1 次元蔵本-Sivashinsky(K-S) 方程式

$$\frac{\partial U}{\partial t} = -\frac{\partial^2 U}{\partial x^2} - \frac{\partial^4 U}{\partial x^4} - \frac{1}{2}\frac{\partial U^2}{\partial x}$$

境界条件: $U(x+L,t) = U(x,t)$

初期条件: $U(x,0) = 16\max(0,$
$\min(x/L, 0.1 - x/L),$
$20(x/L - 0.2)(0.3 - x/L),$
$\min(x/L - 0.6, 0.7 - x/L),$
$\min(x/L - 0.9, 1 - x/L))$

パラメータ: $L = 2\pi/q, q = 0.025$

を擬スペクトル法を用いて下記のように ODE 化したもの [1] を解いた結果を示す。

$$\frac{dy_j}{dt} = ((qj)^2 - (qj)^4)y_j - \frac{\mathrm{i}qj}{2}\mathcal{F}_N(\mathcal{F}_N^{-1}\mathbf{y} \cdot \mathcal{F}_N^{-1}\mathbf{y}) \quad (j = 1, 2, ..., N/2 - 1)$$

ここで $\mathcal{F}_N$, $\mathcal{F}_N^{-1}$ は FFT および逆 FFT を表わす。今回は $N = 1024$ として計算を行う。

2 陰的 Runge-Kutta 法のアルゴリズム

解くべき常微分方程式の初期値問題は下記の通り。

$$\begin{cases} \frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y}) \ \in \mathbb{R}^n \\ \mathbf{y}(t_0) = \mathbf{y}_0 \end{cases} \quad (1)$$

積分区間:$[t_0, \alpha]$

積分区間を $t_0, t_1 := t_0 + h_0, ..., t_{k+1} := t_k + h_k, ...$ と離散化し，t_k における近似解 $\mathbf{y}_k$ から次の t_{k+1} における近似解 $\mathbf{y}_{k+1}$ を求める m 段 IRK 法のアルゴリズムは下記のようになる。

(A) 内部反復: 下記の非線型方程式を未知数 $\mathbf{Y} = [Y_1 \ ... \ Y_m]^T \in \mathbb{R}^{mn}$ について解く。

$$\begin{cases} Y_1 &= \mathbf{y}_k + h_k \sum_{j=1}^m a_{1j}\mathbf{f}(t_k + c_j h_k, Y_j) \\ &\vdots \\ Y_m &= \mathbf{y}_k + h_k \sum_{j=1}^m a_{mj}\mathbf{f}(t_k + c_j h_k, Y_j) \end{cases}$$
$$\Leftrightarrow \mathbf{F}(\mathbf{Y}) = 0 \quad (2)$$

(B) 上記の $\mathbf{Y}$ を用いて次の近似解 $\mathbf{y}_{k+1}$ を求める。

$$\mathbf{y}_{k+1} := \mathbf{y}_k + h_k \sum_{j=1}^m b_j \mathbf{f}(t_k + c_j h_k, Y_j)$$

これに対して，非線型方程式を解く部分に簡易 Newton 法を用い，かつ，連立一次方程式を解く部分に倍精度 (DP)-多倍長精度 (MP) 混合精度反復改良法を使用した結果，1 ステップ進むアルゴリズムは下記のようになる。

初期値: $\mathbf{Y}_{-1} \in \mathbb{R}^{mn}$
For $l = 0, 1, 2, ...$ 簡易 Newton 法
(1) $\mathbf{Y}_l := [Y_1^{(l)} \ Y_2^{(l)} \ ... \ Y_m^{(l)}]^T$
(2) $C := I_m \otimes I_n - h_k X \otimes J$, $||C||_F$ の計算
(3) $\mathbf{d} := (W^T B \otimes I_n)(-\mathbf{F}(\mathbf{Y}_l))$
(4) $C\mathbf{x}_0 = \mathbf{d}$ を $\mathbf{x}_0$ について解く (DP)
For $\nu = 0, 1, 2, ...$ 混合精度反復改良法
(5) $\mathbf{r}_\nu := \mathbf{d} - C\mathbf{x}_\nu$
(6-1) $\mathbf{r}'_\nu := \mathbf{r}_\nu / ||\mathbf{r}_\nu||$ (DP)
(6-2) $C\mathbf{z} = \mathbf{r}'_\nu$ を $\mathbf{z}$ について解く (DP)
(6-3) $\mathbf{x}_{\nu+1} := \mathbf{x}_\nu + ||\mathbf{r}_\nu||\mathbf{z}$
(6-4) 収束判定 $\Rightarrow \mathbf{x}_{\nu_{stop}}$
(7) $\mathbf{Y}_{l+1} := \mathbf{Y}_l + (W \otimes I_n)\mathbf{x}_{\nu_{stop}}$
収束判定 $\Rightarrow \mathbf{Y}_{l_{stop}}$
$\mathbf{Y} := \mathbf{Y}_{l_{stop}} = [Y_1 \ Y_2 \ ... \ Y_m]^T$
$\mathbf{y}_{k+1} := \mathbf{y}_k + h_k \sum_{j=1}^m b_j \mathbf{f}(t_k + c_j h_k, Y_j)$

3 数値実験

Intel Core i7 3820 (4 cores) 3.6GHz + 64GB RAM, Scientific Linux 6.3 x86_64, Intel C++ 13.0.1, MPFR 3.1.1/GMP 5.1.1, BNCpack 0.8 の環境において，OpenMP による 4 スレッド並列化，10 進約 80 桁計算を用いて $t = 10$ における近似値を求めた結果を以下に示す。21〜23 桁の桁落ちが生じていることが分かる。

80 桁計算	$RTOL = ATOL = 10^{-60}$		
段数 (m)	20	30	40
80 計算時間 (s)	130165.4	160601.8	133541.0
ステップ数	6911	2667	1103
最大相対誤差	4.2E-38	2.7E-38	1.4E-38
最小相対誤差	1.1E-54	1.8E-50	1.7E-52
	$RTOL = ATOL = 10^{-70}$		
段数 (m)	20	30	40
計算時間 (s)	100695.2	86331.4	137232.9
ステップ数	6978	1738	1175
最大相対誤差	4.4E-48	1.9E-49	2.4E-47
最小相対誤差	2.7E-68	5.9E-68	1.3E-62

4 今後の課題

今回使用した K-S 方程式の離散化手法の場合，FFT, 逆 FFT を含む被積分関数 $\mathbf{f}(t, \mathbf{y})$ の計算で 8〜9 割の計算時間を占めている。これを GPU もしくは Xeon Phi のようなメニーコア環境を用いてより高速に計算できるよう，ライブラリを整備してチャレンジしたい。

参考文献

[1] E. ハイラー, G. ヴァンナー, 三井斌友・監訳. 常微分方程式の数値解法 II 応用編. シュプリンガー・ジャパン, 2008.
[2] 幸谷智紀. 並列化した多倍長陰的 Runge-Kutta 法の性能分析. 情報処理学会研究報告 HPC, 2013.