

# tk\_ode

---

ODE の数値解析のためのパッケージ群  
1.0 版  
2021 年 6 月 13 日

by Nobuki Takayama

---



## 1 tk\_oder について

この文書は ODE の数値解析を補助するパッケージ群のマニュアルである. パッケージは多くのファイルに分かれており,

```
load("tk_ode.rr")$
```

とするとすべてのパッケージをロードできる.

主なパッケージは単独で読み込める. ODE(ordinary differential equation)を MPFR(任意精度小数パッケージ)を援用して数値解析するための C 言語のプログラムを生成パッケージが tk\_ode\_by\_mpfr である. 下記のコマンドでこのパッケージがロードされる.

```
load("tk_ode_by_mpfr.rr")$
```

このパッケージは HGM に出現する ODE の数値解析のために作成された. HGM については次の web サイトを参照.

- [hgm] <http://www.math.kobe-u.ac.jp/OpenXM/Math/hgm/ref-hgm.html>

内部関数の仕様は man-asir2mpfr.tex 参照(まだ未公開).

## 2 tk\_ode\_by\_mpfr 関数

### 2.1 tk\_ode\_by\_mpfr

`code_solve_ode_by_rk4_with_defuse(Pmat,T0,F0,T1)`  
 :: 係数 $Pmat$ をもつ ODE の初期値問題を解く C 言語のコードを生成する.

`return` C 言語のコード. `main` 関数を含む.

$Pmat$  ODE  $dF/dt = P F$  の係数行列  $P(t)$ .  $t$  の式を成分とすること.

$T0$  初期時刻.

$F0$  初期条件.

$T1$  終了時刻.  $T1 > T0$  を満たすこと.

- $dF/dt = PF$  を  $F(T0)=F0$  の初期条件の元, 時刻  $T1$  まで求める.
- big float による matrix factorial を用いて計算する. 仮数部(significand)のビットサイズは生成したプログラムに `PREC` の値として定義されている. 仮数部のサイズを変更するには生成された C プログラムの `#define PREC 64` の部分を変更するかこの関数の option `prec` を用いる.
- defusing heuristics や知られている値を用いて不安定性を回避コードも含む. 初期値の値がエラーを含む場合, 本来の解でないものがドミナントとなる場合がある. この方法はそれを修正して解くのに有効である. また知られている値が誤差を含む場合も有効である. これらは下記の option 引数でコントロールする.

Option		default value
<code>verbose</code>		0
<code>prec</code>	significand size of MPFR	64
<code>progrname</code>		<code>tmp-test</code>
<code>h</code>	step size	0.001
<code>t_noproj</code>	time to apply defusing	0
<code>n_prune</code>	number of eigen vectors to prune	1
<code>strat</code>	projection strategy	1
<code>n_defuse</code>	number of the matrix factorial	5 $[1/h]$
<code>ref_value_file</code>	File name of exact values	<code>tmp_ref_value.txt</code>

例: Airy の微分方程式  $dF/dt = [[0,1],[t,0]]F$ ,  $F(0) = [0.355028053887817, -0.258819403792807]$  の解  $F(t)$  を  $t=10.1$  まで計算.

```
--> load("tk_ode_by_mpfr.rr");
--> Code=tk_ode_by_mpfr.code_solve_ode_by_rk4_with_defuse([[0,1],[t,0]],
    0,[0.355028053887817,-0.258819403792807],10.1 | h=0.001)$
--> util_write_string_to_a_file("tmp-test.c",Code)$
```

On the unix shell

```
ln -s ${OpenXM_HOME}/lib/asir-contrib/tk_ode_by_mpfr/proj.c tmp-proj.c
cc -I${OpenXM_HOME}/lib/asir-contrib/tk_ode_by_mpfr -DNN=2 -c tmp-proj.c
cc -o tmp-test tmp-test.c tmp-proj.o -lmpfr -lgmp -lgs1 -lgs1cblas -lm
```

```
./tmp-test --verbose --t_noproj 8.1 --n_defuse 2000 --n_prune 1
```

上記の compile 用のコマンドは上記のコマンドが出力する. `--t_noproj 8.1` は  $t < 8.1$  までは 4 次の Runge-Kutta 法を適用する.  $t = 8.1$  以降は `--n_prune` で指定した個数の固有空間 (Re(固有値)の順)の成分を削除する. `--n_defuse 2000` は 2000 個の行列の matrix factorial を計算する.

例:  $H^k_n(1, t)$   $k=10$ ,  $n=1$ ,  $10000 \leq t \leq 10100$  の計算とその値  $f$  と真の値  $H$  との relative error  $(f-H)/H$  の計算. "Hkn10000" は "h2" でもよい(alias).

```
--> load("tk_ode_assert.rr");
--> tk_ode_assert.usage_assert(); // Usage of assert
--> tk_ode_assert.hkn2()
==> Compile tmp-test.c following the instruction.
==> ./tmp-test --go --output_stepsize 1 | grep '^gnuplot' | awk 'print $2,$3' > t.txt
    // Data is stored in t.txt.
--> tk_ode_assert.output_relative_error("Hkn10000", "t.txt" | zoom=10^4301);
    // The initial value for tmp-test is multiplied by 10^4301.
    // Output tmp-rerror.txt
==> Start the gnuplot and plot "tmp-rerror.txt" w lp to show the relative error
```

例: Airy や  $H^k_n(1, t)$  で  $1 \leq t \leq 400$  で同様.

```
--> tk_ode_assert.airy1()
==> Run tmp-test
--> tk_ode_assert.output_relative_error("a", "t.txt" | zoom=1);

--> tk_ode_assert.hkn1()
==> Run tmp-test
--> tk_ode_assert.output_relative_error("h", "t.txt" | zoom=1);
```

## 参照

### ChangeLog

- tk\_ode\_by\_mpfr/tk\_man2mpfr.rr

### 3 tk\_ode\_sparse\_interp 函数

# Index

(インデックスがありません)

(インデックスがありません)

## 簡単な目次

1	tk_oder について .....	1
2	tk_ode_by_mpfir 関数 .....	2
3	tk_ode_sparse_interp 関数 .....	4
	Index .....	5



## 目次

<b>1</b>	<b>tk_oder について</b> .....	<b>1</b>
<b>2</b>	<b>tk_ode_by_mpfr 関数</b> .....	<b>2</b>
2.1	tk_ode_by_mpfr.....	2
<b>3</b>	<b>tk_ode_sparse_interp 関数</b> .....	<b>4</b>
	<b>Index</b> .....	<b>5</b>

